BAB 2
LANDASAN TEORI

2.1 Penjadwalan

Penjadwalan adalah pengalokasian sumber daya pada objek-objek yang ada terhadap ruang waktu yang tersedia sedemikian rupa sehingga sedapat mungkin memenuhi kondisi-kondisi tertentu [8]. Dalam proses penjadwalan, sumber daya yang ada harus dialokasikan secara optimal, efektif dan efisien, serta sedapat mungkin memenuhi kebutuhan yang ada. Fang [9] berpendapat bahwa penjadwalan merupakan masalah yang menantang di bidang *Artificial Inteligence*, hal ini disebabkan oleh beberapa hal yang di antaranya adalah:

1. Penjadwalan merupakan masalah *NP-Complete (Non-deterministic Polynomial Complete)*, yaitu suatu permasalahan dimana dengan bertambahnya *variable* secara linear maka waktu yang diperlukan untuk memecahkan masalah itu bertambah secara ekponensial. Hal ini sering disebut juga *Combinatorial Explosion*, selain itu ruang solusi yang mungkin (*space of possible solutions*) untuk kebanyakan masalah riil terlalu besar untuk dilakukan *brute-force search* atau pencarian yang tidak terarah (*undirected search*).
2. Teknik pencarian tingkat tinggi dengan menggunakan berbagai *heuristic* untuk memangkas *search space* tidak menjamin dihasilkannya suatu solusi yang optimal dan sangat sulit untuk mendesain suatu *heuristic* yang efektif.
3. Masalah penjadwalan sering menjadi rumit karena adanya *special constraints*.
4. Penjadwalan di dunia nyata sering melibatkan *constraint* yang tidak dapat dipresentasikan atau bahkan tidak dapat dinyatakan dengan tepat.

2.1.1 *Constraints Penjadwalan*

Di dalam penjadwalan dikenal dua jenis *constraint* (persyaratan), yaitu *hard constraints* dan *soft constraints*. *Hard constraints* adalah *constraint* yang tidak boleh dilanggar sama sekali sedangkan *soft constraints* adalah *constraints* yang

sedapat mungkin dipenuhi. Pelanggaran terhadap satu *hard constraints* saja akan menyebabkan jadwal tersebut menjadi tidak berguna sedangkan pelanggaran terhadap *soft constraints* masih bisa diterima namun akan menimbulkan ketidaknyamanan pihak yang memakai jadwal tersebut [9].

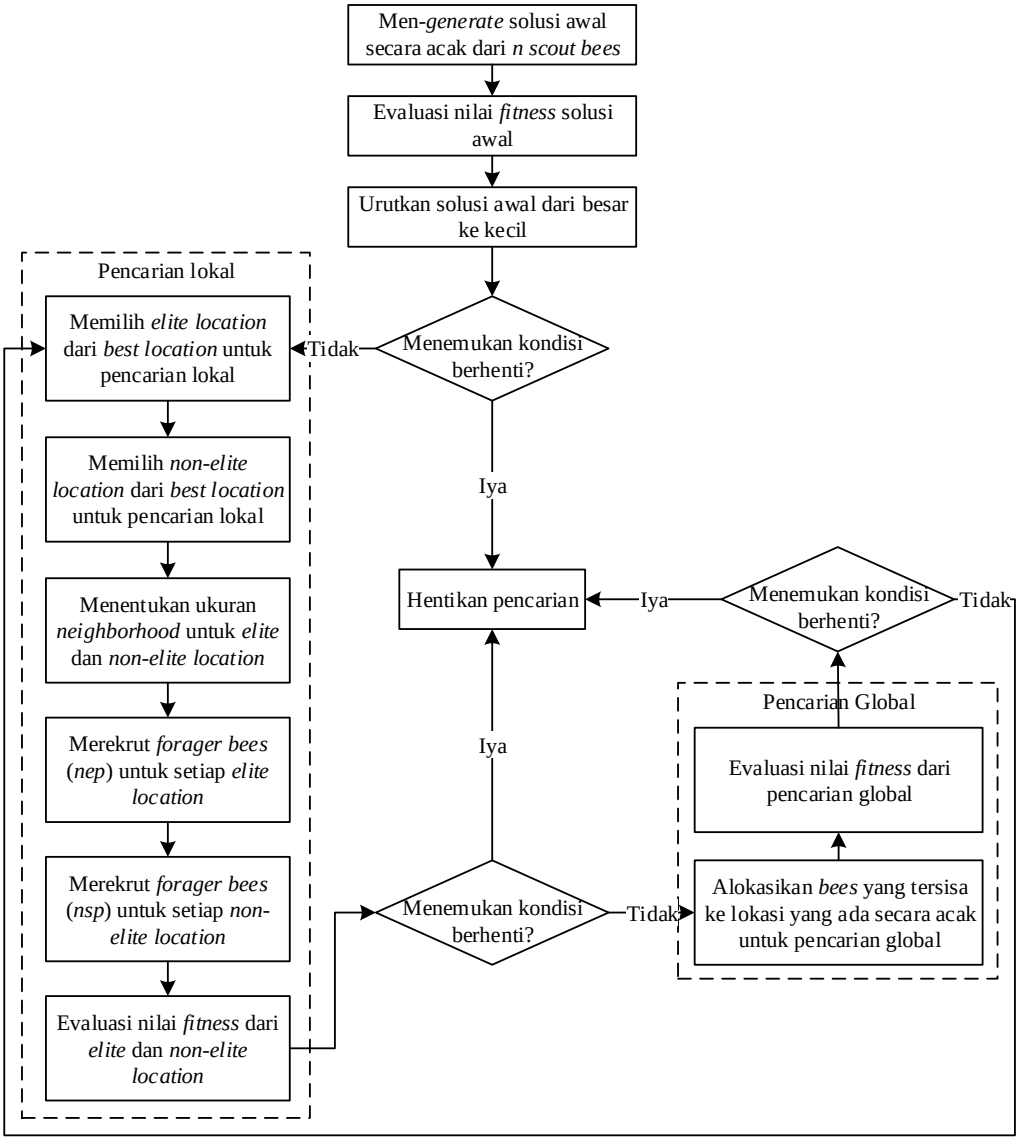
2.2 Algoritma

Menurut Munir [10] algoritma adalah urutan langkah-langkah untuk memecahkan suatu masalah. Algoritma adalah deretan instruksi yang jelas untuk memecahkan masalah, yaitu untuk memperoleh keluaran yang diinginkan dari suatu masukan dalam jumlah waktu yang terbatas [11]. Algoritma adalah prosedur komputasi yang terdefinisi dengan baik yang menggunakan beberapa nilai sebagai masukan dan menghasilkan beberapa nilai yang disebut keluaran [12].

2.3 Algoritma *Bees*

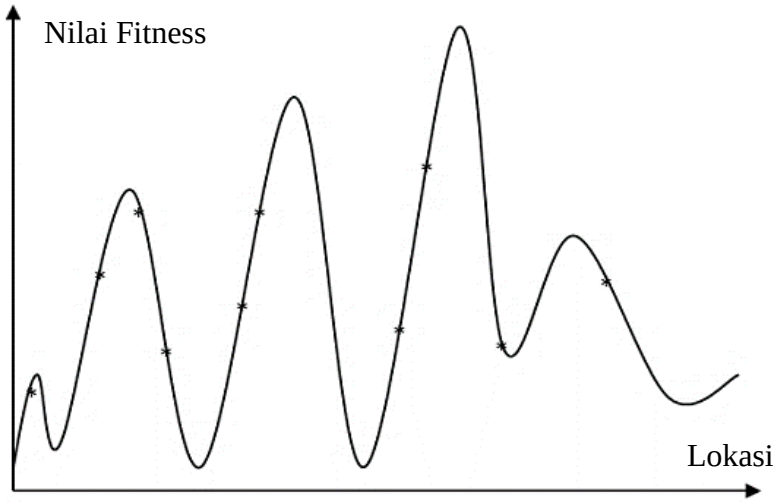
Menurut Pham [2] algoritma *Bees* adalah algoritma yang termasuk dalam bidang algoritma *Swarm Intelligence*, algoritma ini masih berhubungan dengan algoritma lain yang terinspirasi dari lebah seperti algoritma *Artificial Bee Colony*, dan algoritma *Swarm Intelligence* lain seperti *Ant Colony Optimization* dan *Particle Swarm Optimization*. Algoritma *Bees* terinspirasi dari perilaku lebah madu saat mencari makan. Lebah madu mengumpulkan nektar dari area yang luas di sekitar sarang Mereka. Suatu koloni mengirimkan sejumlah lebah untuk mengumpulkan nektar dari *patch* bunga. Lebah saling berkomunikasi menggunakan *waggle dance* untuk menginformasikan arah, jarak, dan penilaian kualitas dari sumber makanan kepada lebah lain di sarang. Tujuan pengolahan informasi dari algoritma *Bees* adalah untuk mencari dan mengeksplorasi *best patch* dalam ruang pencarian masalah. *Scout bee* dikirim keluar untuk mencoba ruang masalah secara acak dan menemukan *best patch*. *Best patch* dieksploitasi menggunakan *local search*, di mana sejumlah *best patch* akan lebih dieksplorasi dari yang lain. *Best patch* akan terus dieksploitasi, meskipun banyak *scout bee* dikirim keluar setiap iterasi untuk mencari *best patch* baru.

Algoritma *Bees* mirip dengan algoritma *Artificial Bee Colony* dalam artian memiliki proses pencarian lokal dan pencarian global. Namun ada perbedaan di antara kedua algoritma tersebut selama proses pencarian lokal. Algoritma *Artificial Bee Colony* menggunakan pendekatan probabilistik selama tahap pencarian lokal, sedangkan algoritma *Bees* tidak menggunakan pendekatan probabilistik, melainkan menggunakan evaluasi nilai *fitness* untuk mendorong pencariannya [3]. Alur algoritma Bees dapat dilihat pada Gambar 2.1.



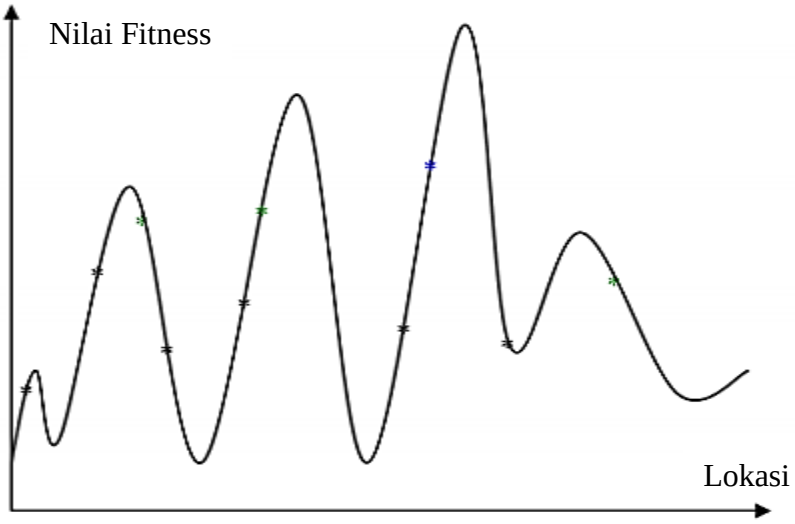
Gambar 2.1 Alur algoritma Bees

Algoritma *Bees* dimulai dengan menempatkan sejumlah (*n*) *scout bees* secara acak pada *search space* (Gambar 2.2).



Gambar 2.2 Visualisasi sejumlah *n scout bees* pada *search space*

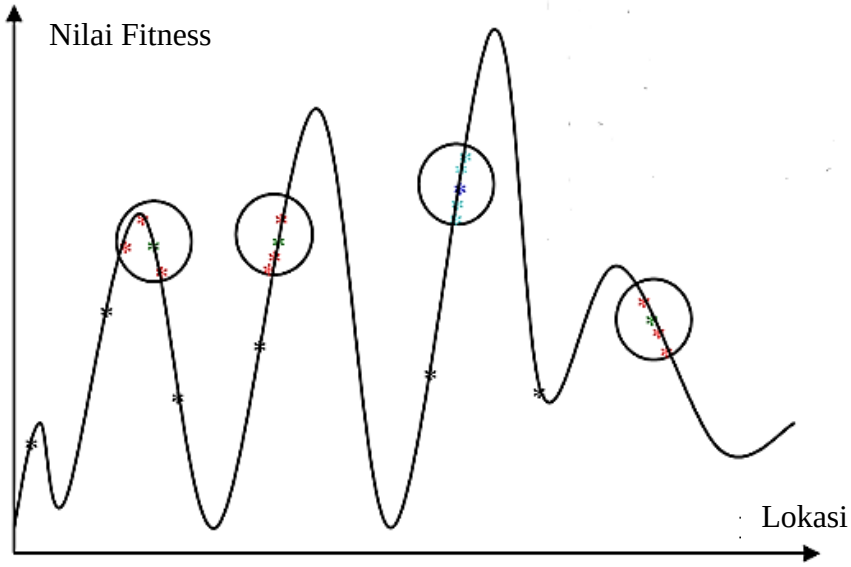
Kemudian nilai *fitness* dari lokasi yang dikunjungi oleh *scout bees* akan dievaluasi dan diurutkan dari besar ke kecil. Proses pencarian lokai akan menggunakan lokasi terbaik, yaitu sejumlah (*m*) *fittest location*s. Lokasi terbaik (*m*) akan dibagi menjadi dua grup, yaitu lokasi terbaik *elite* (*e*) dan lokasi terbaik *non elite* (*m-e*) (Gambar 2.3).



Gambar 2.3 Visualisasi pemilihan *elite* dan *non-elite bees*

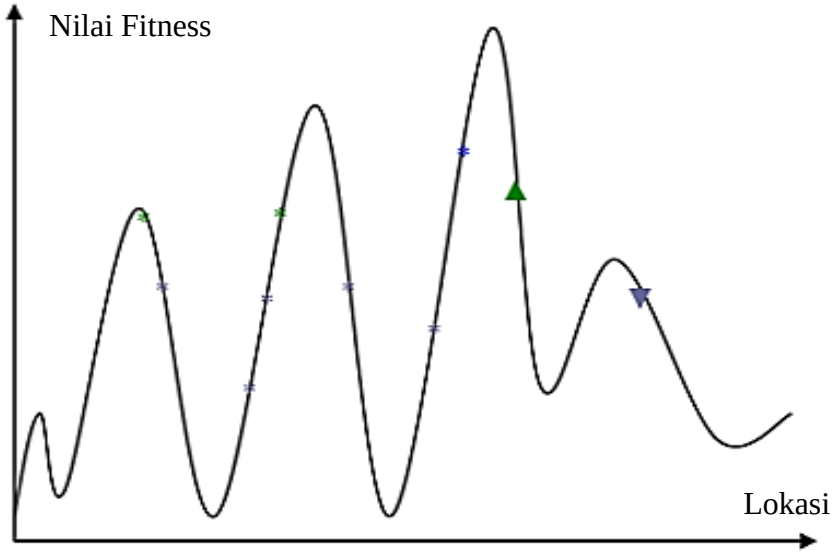
Proses *local search* dimulai dengan merekrut *bees* di sekitar lokasi terbaik. Parameter untuk ukuran pencarian adalah (*ngh*), jumlah lebah yang direkrut di

sekitar lokasi *elite* adalah (*nep*), dan jumlah lebah yang direkrut di sekitar lokasi non elite adalah (*nsp*) (Gambar 2.4).



Gambar 2.4 Visualisasi perekrutan *bees* pada *elite* dan *non-elite location*

Proses global search adalah proses pencarian random pada lokasi *non-best* (*n-m*) (Gambar 2.5).



Gambar 2.5 Visualisasi hasil local search dan global search

Tahap terakhir adalah mengurutkan berdasarkan nilai *fitness* dan proses akan terus berlangsung sampai *global optimum* ditemukan.

Algoritma Bees membutuhkan beberapa parameter, parameter tersebut adalah sebagai berikut.

Tabel 2.1 Parameter algoritma Bees

Parameter	Simbol
Jumlah *scout bees* pada *patches* yang terpilih	*n*
Jumlah *patches* terbaik dari *patches* yang terpilih	*m*
Jumlah *elite patches* pada *patches* terbaik yang terpilih	*e*
Jumlah *bees* yang direkrut pada *elite patches*	*nep*
Jumlah *bees* yang direkrut pada *non-elite patches* terbaik	*nsp*
Ukuran pencarian untuk setiap *patch*	*ngh*

2.4 Algoritma *Harmony Search*

Algoritma *Harmony Search* adalah algoritma yang terinspirasi dari proses improvisasi musisi, secara spesifik yaitu pada saat para musisi (yang mungkin sebelumnya belum pernah bermain bersama) dengan cepat berimprovisasi untuk menghasilkan *harmony* yang sesuai perkiraan estetika. Dengan menggunakan analogi tersebut, maka algoritma *Harmony Search* melakukan proses optimasi untuk mendapatkan keadaan terbaik dengan cara melakukan evaluasi terhadap nilai *fitness*. Seperti halnya perkiraan estetika yang ditentukan menggunakan himpunan *pitches* yang dikeluarkan alat musik, nilai *fitness* pada algoritma *Harmony Search* dihitung menggunakan himpunan nilai-nilai pada setiap variabel keputusan. Perbaikan nilai *fitness* pada algoritma *Harmony Search* menerapkan improvisasi yang terus ditingkatkan dari tiap iterasi ke iterasi, sama seperti perbaikan kualitas estetika yang diperbaiki dengan cara melakukan latihan demi latihan. Jika *harmony* baru yang diciptakan lebih baik dibandingkan dengan *harmony* sebelumnya, maka *harmony* baru ini akan tetap dipertahankan. Perbandingan dari proses improvisasi *harmony* dan optimasi dapat dilihat pada Tabel 2.2.

Tabel 2.2 Perbandingan Improvisasi *Harmony* dan Optimasi

Faktor Perbandingan	Improvisasi *Harmony*	Optimasi
Target	Standar estetika	Nilai *fitness*
Keadaan terbaik	*Fantastic harmony*	*Global Optimum*
Komponen	*Pitches* dari instrumen	Nilai dari variabel
Unit proses	Setiap latihan	Setiap iterasi

Improvisasi musik adalah proses untuk mencari *harmony* yang lebih baik dengan mencoba berbagai kombinasi *pitches* yang harus mengikuti tiga aturan, yaitu:

1. Memainkan satu *pitch* apapun yang ada di *memory*.
2. Memainkan satu *pitch* yang berdekatan dengan *pitch* yang ada di *memory*.
3. Memainkan *pitch* acak dari rentang nada yang memungkinkan.

Aturan pada proses ini ditiru pada algoritma Harmony Search sebagai berikut:

1. Memilih satu nilai dari *harmony memory*.
2. Memilih satu nilai yang berdekatan dengan nilai yang ada di *harmony memory*.
3. Memilih nilai acak dari rentang nilai yang memungkinkan.

Tiga aturan ini secara efektif diatur menggunakan dua parameter, yaitu *harmony memory consider rate* (*HMCR*) dan *pitch adjusting rate* (*PAR*).

Berdasarkan konsep tersebut, secara garis besar algoritma *Harmony Search* terdiri dari lima tahapan, yaitu:

1. Identifikasi masalah dan parameter algoritma *Harmony Search*, dengan parameter yang dibutuhkan adalah sebagai berikut.

Tabel 2.3 Parameter algoritma Harmony Search

Parameter	Simbol
Harmony Memory Size, adalah banyaknya vektor solusi yang terbentuk	HMS
Harmony Memory Consider Rate, adalah probabilitas pemilihan komponen dari Harmony Memory yang ada, bernilai $0 \leq \text{HMCR} \leq 1$. Umumnya berkisar antara 0.7 sampai 0.99	HMCR
Pitch Adjustment Rate, adalah probabilitas suatu anggota Harmony Memory untuk dimutasi bernilai $0 \leq \text{PAR} \leq 1$. Umumnya berkisar antara 0.1 sampai 0.5.	PAR

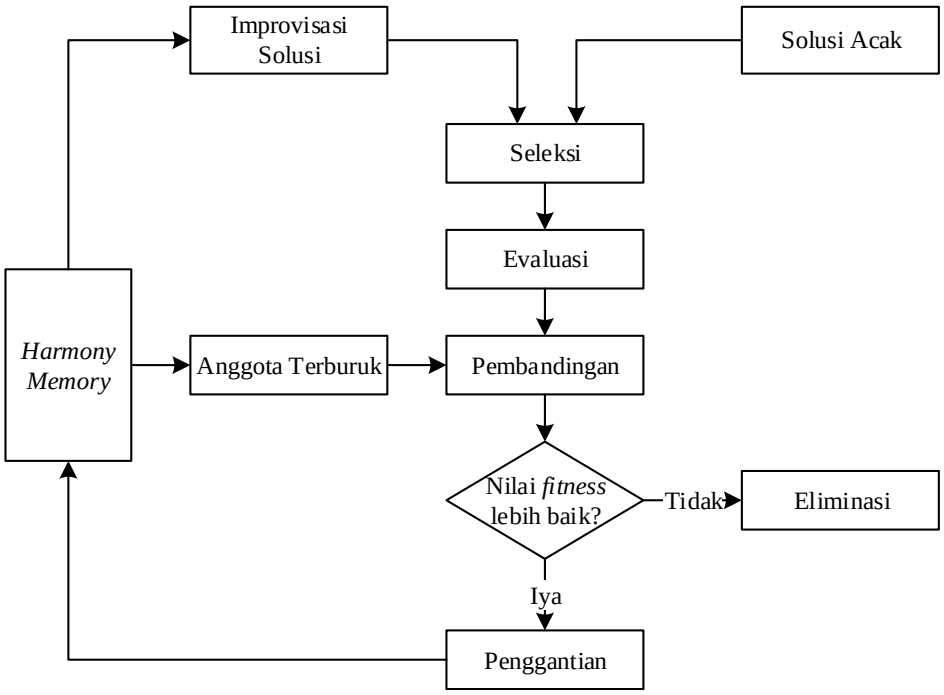
2. Inisialisasi *Harmony Memory*, *harmony memory* awal didapatkan dari membangkitkan solusi secara acak, untuk masalah sebesar *n*-dimensi dapat direpresentasikan sebagai berikut.

$$\mathrm{HM} = \begin{bmatrix} x_1^1, x_2^1, \ldots, x_n^1 \\ x_1^2, x_2^2, \ldots, x_n^2 \\ \vdots \\ x_1^{\mathrm{HMS}}, x_2^{\mathrm{HMS}}, \ldots, x_n^{\mathrm{HMS}} \end{bmatrix} \quad (2.1)$$

Di mana $[x_{1,}^i x_2^i, \ldots, x_n^i]$ (i=1,2,…,HMS) adalah kandidat solusi.

3. Improvisasi *Harmony* Baru $[x_{1,}^j x_2^j, \ldots, x_n^j]$. Setiap komponen dari solusi ini didapatkan berdasarkan HMCR. HMCR didefinisikan sebagai probabilitas pemilihan komponen dari *Harmony Memory* yang ada. Jika x_1^j diambil dari *Harmony Memory* maka dipilih secara acak dari dimensi ke-j anggota *Harmony Memory*, dan dapat dimutasikan berdasarkan PAR. PAR adalah probabilitas suatu anggota *Harmony Memory* untuk dimutasi. Proses improvisasi ini mirip dengan proses pembuatan *offspring* pada algoritma genetika dengan proses mutasi dan *crossover*.
4. Memperbarui *Harmony Memory*. Solusi dari tahap ketiga akan dievaluasi, jika nilai *fitness* dari vektor solusi baru nilainya lebih baik daripada vektor terburuk yang ada pada *Harmony Memory*, maka vektor solusi baru akan digunakan, sebaliknya jika lebih buruk maka akan dieliminasi.
5. Kriteria berhenti, menentukan banyanya proses pembangkitan solusi baru dan memperbarui *Harmony Memory*. Apabila kriteria berhenti terpenuhi maka proses iterasi akan berhenti.

Algoritma *Harmony Search* menggunakan teknik pencarian acak, seperti algoritma *Genetic*, *Particle Swarm Optimization*, dan *Differential Evolution.* Algoritma *Harmony Search* memiliki karakteristik yang sederhana, dan memiliki kekurangan pada pencarian lokal yang lemah. Alur algoritma *Harmony Search* dapat dilihat pada Gambar 2.6.



Gambar 2.6 Alur algoritma *Harmony Search*

2.5 *Object-oriented Programming (OOP)*

Object-oriented programming adalah sebuah pendekatan untuk pengembangan perangkat lunak di mana struktur perangkat lunak didasarkan pada interaksi suatu objek dengan objek lain untuk menyelesaikan tugas. Interaksi ini berbentuk pesan yang dikirim bolak-balik di antara objek. Dalam menanggapi suatu pesan, sebuah objek dapat melakukan tindakan [13]. Konsep dasar dari OOP adalah sebagai berikut:

1. *Object*, merupakan kumpulan dari variabel dan fungsi yang saling berkaitan.
2. *Class*, merupakan suatu rancangan dari objek dan dapat berisi banyak objek.
3. *Encapsulation*, merupakan proses di mana permintaan akses suatu data tidak dapat dilakukan secara langsung.
4. *Polymorphism*, merupakan kemampuan dua objek yang berbeda unguk merespon pesan permintaan yang sama menggunakan caranya masing-masing.

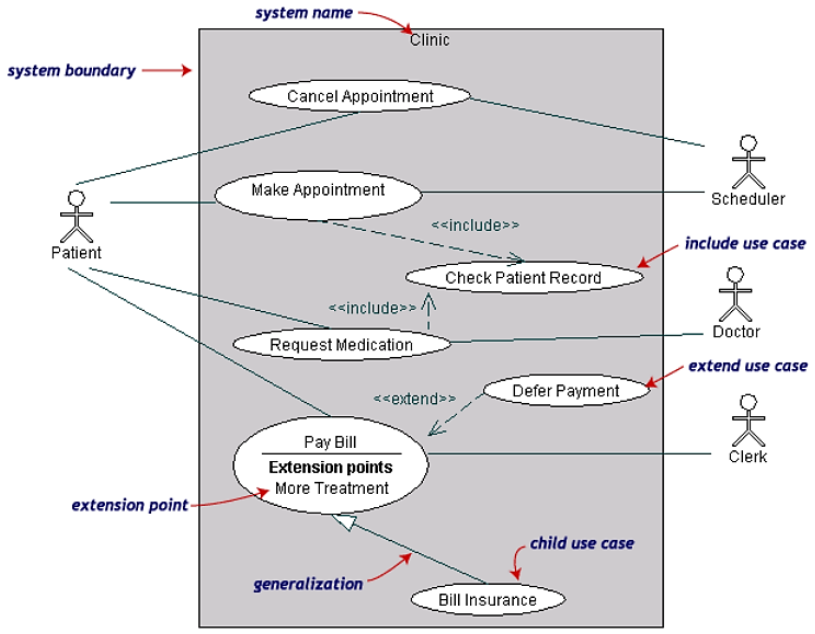
5. *Inheritance*, merupakan kemampuan untuk memperluas suatu *class* dengan mewarisi semua karakter dari *class* induknya.

2.6 *Unified Modelling Language*

Unified Modelling Language (UML) adalah sebuah bahasa yang telah menjadi standar dalam industri untuk memvisualisasi, merancang dan mendokumentasikan sistem piranti lunak. UML menawarkan sebuah standar untuk merancang model sebuah sistem. 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 tiga notasi yang telah ada sebelumnya, yaitu Grady Booch OOD *(Object-Oriented Design)*, Jim Rumbaugh OMT *(Object Modeling Technique)*, dan Ivar Jacobson OOSE *(Object-Oriented Software Engineering)* [14].

2.6.1 *Usecase Diagram*

Usecase diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah apa yang diperbuat sistem, dan bukan bagaimana. Sebuah *usecase* merepresentasikan sebuah interaksi antara aktor dengan sistem. *Usecase* merupakan sebuah pekerjaan tertentu, misalnya login ke sistem, membuat sebuah daftar belanja, dan sebagainya. Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu. *Usecase diagram* dapat sangat membantu saat menyusun *requirement* sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang *test case* untuk semua *feature* yang ada pada sistem.



Gambar 2.7 Usecase Diagram

2.6.2 *Class Diagram*

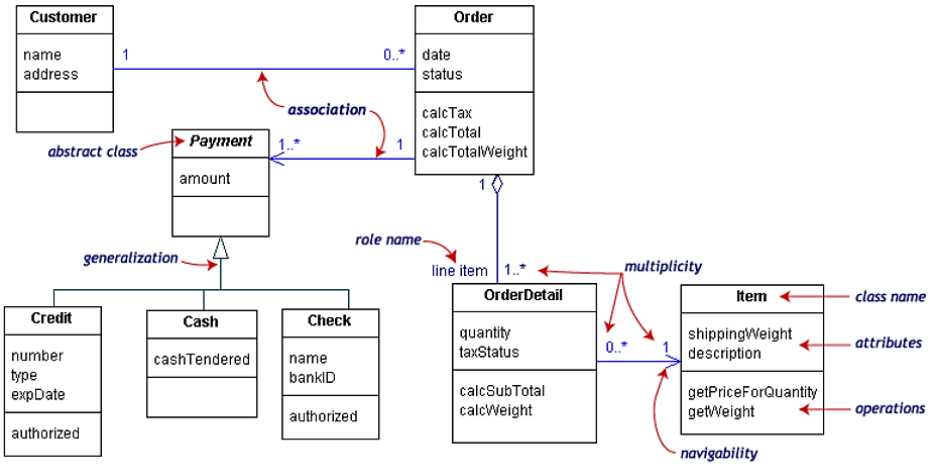
Class adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. *Class* menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metode/fungsi). *Class diagram* menggambarkan struktur dan deskripsi *class*, *package* dan objek beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain.

Class memiliki tiga area pokok, yaitu:

1. Nama
2. Atribut
3. Metode

Atribut dan metode dapat memiliki salah satu sifat berikut:

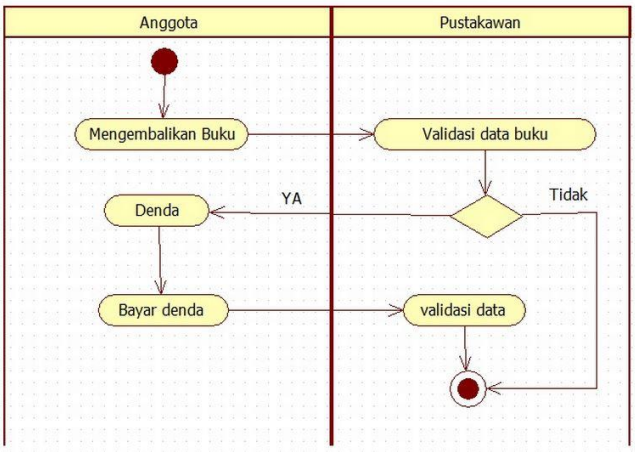
1. *Private*, tidak dapat dipanggil dari luar class yang bersangkutan.
2. *Protected*, hanya dapat dipanggil oleh class yang bersangkutan dan anak-anak yang mewarisinya.
3. *Public*, dapat dipanggil oleh siapa saja.



Gambar 2.8 Class Diagram

2.6.3 *Activity Diagram*

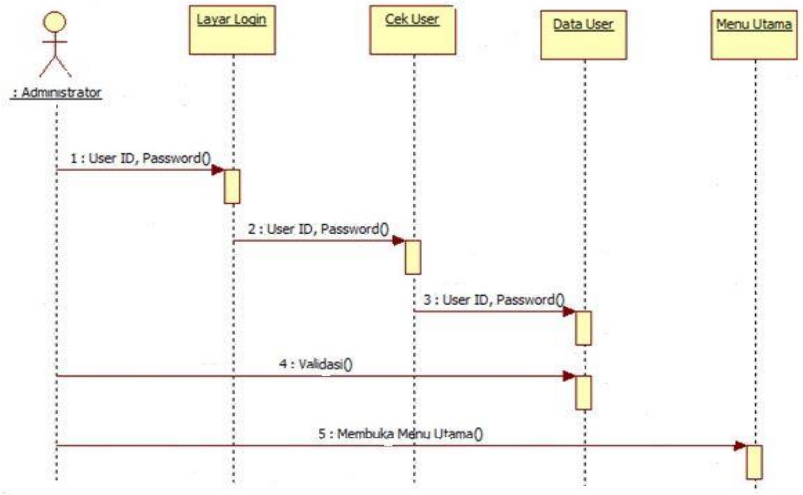
Activity diagram menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana berakhirnya. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi. *Activity*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



Gambar 2.9 *Activity* Diagram

2.6.4 *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 atar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait). *Sequence diagram* biasa digunakan untuk menggambarkan skenario atau rangkaian 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.



Gambar 2.10 Sequence Diagram

2.7 Basis Data

Basis data adalah kumpulan data yang disimpan secara sistematis di dalam komputer dan dapat diolah atau dimanipulasi menggunakan perangkat lunak (program aplikasi) untuk menghasilkan informasi [15]. Basis data adalah kumpulan data yang terhubung secara logika yang bisa dipakai secara bersama dan deskripsi mengenai data tersebut, yang didesain untuk memenuhi kebutuhan informasi dari sebuah organisasi. Berdasarkan pendapat di atas disimpulkan bahwa basis data adalah sekumpulan data yang saling berhubungan dan merupakan aspek yang

sangat penting dalam sistem informasi di mana basis data merupakan gudang penyimpanan data yang akan diolah lebih lanjut.

2.8 *Tools* Pembangunan Sistem

Tools pembangunan sistem yang digunakan termasuk bahasa pemrograman yang digunakan dan perangkat lunak yang digunakan untuk membangun sistem.

2.8.1 *C-Sharp (C#)*

C-*Sharp* merupakan sebuah bahasa pemrograman yang berorientasi objek yang dikembangkan oleh Microsoft sebagai bagian dari inisiatif kerangka *.NET Framework*. Bahasa pemrograman ini dibuat berbasiskan bahasa C++ yang telah dipengaruhi oleh aspek-aspek ataupun fitur bahasa yang terdapat pada bahasa-bahasa pemrograman lainnya seperti *Java*, *Delphi*, *Visual Basic*, dan lain-lain dengan beberapa penyederhanaan [15]. Bahasa C# secara teknis mengadopsi sintaks bahasa C/C++. Konsistensi API membuat bahasa C# menjadi pilihan dalam membuat kode pemrograman diatas *platform* Windows. Bahasa C# mendukung object *oriented programming* dan juga *dymanics programming*. Ini menambahkan daftar kemudahan untuk belajar bahasa C#. Bahasa C# ditujukan agar cocok digunakan untuk menulis program aplikasi baik dalam *hosted system* maupun *embedded system*, mulai dari program aplikasi yang sangat besar yang menggunakan sistem operasi yang canggih hingga kepada program aplikasi yang sangat kecil yang memiliki fungsi-fungsi terdedikasi. Tujuan bahasa C# menurut Hejilberg

2.8.2 *Microsoft Visual Studio*

Microsoft Visual Studio merupakan sebuah perangkat lunak lengkap (*suite*) yang dapat digunakan untuk melakukan pengembangan aplikasi, baik itu aplikasi bisnis, aplikasi perorangan, ataupun komponen aplikasinya, dalam bentuk aplikasi *console*, aplikasi Windows, ataupun aplikasi *web*. *Visual Studio* mencakup kompiler, SDK, *Integrated Development Environment (IDE)*, dan dokumentasi (umumnya berupa *MSDN Library*). *Microsoft Visual Studio* dapat digunakan untuk

mengembangkan aplikasi dalam *native code* (dalam bentuk bahasa mesin yang berjalan di atas Windows) ataupun *managed code* (dalam bentuk Microsoft Intermediate Language di atas .NET Framework). Selain itu, *Visual Studio* juga dapat digunakan untuk mengembangkan aplikasi *Silverlight*, aplikasi Windows Mobile (yang berjalan di atas .NET Compact Framework).

2.8.3 *MySQL*

Menurut Sutarman, MySQL dikembangkan oleh sebuah perusahaan Swedia bernama MySQL AB, yang pada saat itu bernama TcX DataKonsult AB, sejak sekitar 1994-1995, meski cikal bakal kodenya bisa di sebut sudah ada sejak 1979. Tujuan mula-mula TcX membuat MySQL pada waktu itu juga memang untuk mengembangkan aplikasi *web* untuk klien. TcX adalah perusahaan pengembang *software* dan konsultan *database*. Pada saat itu Michael Widenius atau "Monty", pengembang satu-satunya di TcX, memiliki aplikasi UNIREG dan rutin ISAM yang dibuat sendiri dan sedang mencari antar muka SQL untuk ditempelkan diatasnya.

2.9 Teknik Pengujian Sistem

Pengujian sistem dilakukan pada saat sistem telah selesai dibuat, tujuan pengujian ini adalah untuk mengetes kualitas dari sistem yang telah dibuat. Terdapat 4 jenis pengujian, yaitu Pengujian Statis, Pengujian *Black box*, Pengujian *White box* dan Pengujian performansi [16]. Dalam Penelitian ini digunakan Pengujian *Black box* dan Pengujian Performansi.

2.9.1 Pengujian Performansi

Pengujian performansi merupakan pengujian yang dilakukan apabila perangkat lunak telah berjalan dengan benar. Pengujian ini dilakukan bukan untuk mengetes kebenaran dari sistem namun hasil dan waktu respon dari perangkat lunak. Pengujian performansi dilakukan dengan menyiapkan peralatan test. Pengujian akan menguji sistem mulai dari saat tidak bekerja hingga saat sedang puncak sibuk bekerja. Pengujian performansi ini berbeda dengan pengujian *white box* ataupun *black box*, apabila kecacatan ditemukan oleh pengujian *white box* atau

black box maka akan dilakukan koreksi program, namun pengujian performansi akan lebih memeriksa aplikasi untuk kemampuannya terhadap *hardware*, dimana kecacacatan dalam pengujian performansi akan membuat developer menyarankan pembelian *hardware* yang lebih memadai untuk sistemnya [16].

2.9.2 Pengujian *Black box*

Pengujian *Black box* berfokus pada persyaratan fungsional perangkat lunak. Pengujian *Black box* memungkinkan perekayasa perangkat lunak mendapatkan serangkaian kondisi input yang sepenuhnya menggunakan semua persyaratan fungsional untuk suatu program [17].

Pengujian *Black box* adalah pengujian aspek fundamental sistem tanpa memperhatikan struktur logika internal perangkat lunak. Metode ini digunakan untuk mengetahui apakah perangkat lunak berfungsi dengan benar. *Black box* merupakan metode perancangan data uji yang didasarkan pada spesifikasi perangkat lunak. Data uji dieksekusi pada perangkat lunak dan kemudian keluar dari perangkat lunak dicek apakah telah sesuai yang diharapkan.

Pengujian *Black box* berusaha menemukan kesalahan dalam kategori sebagai berikut.

1. Fungsi-fungsi yang tidak benar atau hilang
2. Kesalahan *interface*
3. Kesalahan dalam struktur data atau akses *database* eksternal
4. Kesalahan kinerja
5. Inisialisasi dan kesalahan terminasi