PENGEMBANGAN *LIBRARY* ALGORITMA GENETIKA DENGAN PENDEKATAN *OBJECT-ORIENTED DESIGN* DAN *OBJECT-ORIENTED PROGRAMMING*

SKRIPSI

Diajukan untuk memenuhi persyaratan
memperoleh gelar Sarjana Teknik

Disusun Oleh :
EKA PRAKARSA MANDYARTHA
NIM. 0610630032 - 63

KEMENTERIAN PENDIDIKAN DAN KEBUDAYAAN
UNIVERSITAS BRAWIJAYA
FAKULTAS TEKNIK
MALANG
2012

LEMBAR PERSETUJUAN

PENGEMBANGAN *LIBRARY* ALGORITMA GENETIKA DENGAN PENDEKATAN *OBJECT-ORIENTED DESIGN* DAN *OBJECT-ORIENTED PROGRAMMING*

SKRIPSI

Diajukan untuk memenuhi persyaratan
memperoleh gelar Sarjana Teknik

Disusun oleh :

EKA PRAKARSA MANDYARTHA
0610630032 - 63

Telah diperiksa dan disetujui oleh :

Dosen Pembimbing I	Dosen Pembimbing II
Hadi Suyono, ST., MT., Ph.D.	**Adharul Muttaqin, ST., MT.**
NIP. 19730520 200801 1 013	**NIP. 19760121 200501 1 001**

PENGANTAR

Dengan nama Allah SWT Yang Maha Pengasih dan Penyayang. Segala puji bagi Allah SWT karena atas Taufik, Hidayah, Ridho serta InayahNya-lah sehingga penulis dapat menyelesaikan Tugas Akhir dengan judul "**Pengembangan *Library* Algoritma Genetika Dengan Pendekatan *Object-Oriented Design* dan *Object-Oriented Programming***". Tugas Akhir ini disusun untuk memenuhi sebagian persyaratan memperoleh gelar Sarjana Teknik di Jurusan Teknik Elektro Program Studi Rekayasa Komputer Fakultas Teknik Universitas Brawijaya Malang.

Tidak banyak yang bisa penulis sampaikan kecuali ungkapan terima kasih kepada berbagai pihak yang telah dengan tulus ikhlas memberikan bimbingan, arahan, dan dukungan hingga penulisan tugas akhir ini dapat terselesaikan. Pada kesempatan kali ini, penulis mengucapkan banyak terima kasih kepada:

1. Ayahanda dan Ibunda tercinta, juga adikku tercinta Ditya, serta seluruh keluarga yang senantiasa memberikan semangat, doa dan restu demi terselesaikannya skripsi ini.
2. Bapak Sholeh Hadi Pramono, DR., Ir., MS. selaku Ketua Jurusan Teknik Elektro, Fakultas Teknik Universitas Brawijaya.
3. Bapak M. Azis Muslim, ST., MT., Ph.D selaku Sekretaris Jurusan Teknik Elektro, Fakultas Teknik Universitas Brawijaya.
4. Bapak Moch. Rif'an. ST., MT. selaku Ketua Program Studi Jurusan Teknik Elektro, Fakultas Teknik Universitas Brawijaya.
5. Bapak Waru Djuriatno, ST. MT. selaku Ketua Kelompok Dosen Keahlian Rekayasa Komputer Jurusan Teknik Elektro Universitas Brawijaya.
6. Bapak Hadi Suyono, ST., MT., Ph.D. selaku dosen pembimbing I yang telah banyak memberikan bimbingan, masukan dan arahan dalam penyusunan tugas akhir ini.
7. Bapak Adharul Muttaqin, ST., MT. selaku dosen pembimbing II yang telah bersedia memberikan bimbingan, masukan dan arahan dalam penyusunan tugas akhir ini.

8. Bapak dan Ibu Dosen, Staff Administrasi Jurusan Teknik Elektro Fakultas Teknik Universitas Brawijaya.
9. Tomi Putro Utomo yang telah bersedia memberi ilmunya seputar .NET Framework dan C#.
10. Didit, Norman, Adityo, Tito, Abdur, Jumadil, Felix, mas Diriga, mas Prima, mas Runi, Krisna, juga semua teman-teman asisten, serta Ka. Lab dan Laboran Laboratorium Komputasi dan Jaringan yang telah memberikan banyak bantuan dan dukungan selama ini.
11. Saudara Angga, Aflah, Niluh, dan mas Agung atas bantuan dan dukungan dalam menyelesaikan tugas akhir ini.
12. Teman-teman angkatan 2006 (Ge-Force), khususnya paket E 06 dan anggota RisTIE 2009-2010.
13. Semua pihak yang tidak dapat penulis sebutkan satu per satu yang terlibat baik secara langsung maupun tidak langsung demi terselesaikannya tugas akhir ini.

Penulis menyadari sepenuhnya bahwa Tugas Akhir ini masih banyak kekurangan. Segala kritik dan saran yang membangun akan penulis terima demi kesempurnaan skripsi ini. Harapan penulis semoga skripsi ini bermanfaat bagi pembaca dan khususnya mahasiswa jurusan teknik elektro dimasa yang akan datang.

Malang, Februari 2012

Penulis

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

ABSTRAK

EKA PRAKARSA MANDYARTHA 2012. : Pengembangan *Library* Algoritma Genetika Dengan Pendekatan *Object-Oriented Design* dan *Object-Oriented Programming*. Skripsi Jurusan Teknik Elektro, Fakultas Teknik, Universitas Brawijaya. Dosen Pembimbing: Hadi Suyono, ST., MT., Ph.D. dan Adharul Muttaqin, ST., MT.

Algoritma genetika merupakan pendekatan komputasional untuk menyelesaikan suatu masalah dengan memodelkan masalah menjadi proses evolusi biologis. Pemilihan solusi yang akan digunakan untuk membentuk populasi yang baru tergantung nilai *fitness*nya. Pembentukan populasi baru untuk pencarian solusi dilakukan berulang kali hingga memenuhi kondisi tertentu untuk berhenti. *Library* Algoritma Genetika (AG3NLib) merupakan perangkat lunak yang digunakan untuk membantu penyelesaian masalah menggunakan algoritma genetika. Perancangan *library* ini dilakukan dengan pendekatan *object-oriented design.* Implementasi *library* ini menggunakan *object-oriented programming* dengan bahasa pemrograman C#.

Pengujian *library* ini dilakukan untuk memastikan *library* berjalan sesuai dengan fungsi yang ditentukan dan benar perhitungannya. Pengujian dilakukan dengan membandingkan perhitungan AG3NLib dengan hasil menggunakan metode konvensional pada kasus uji tertentu. Pada kasus uji 1 didapatkan nilai *fitness* terbaik yang konsisten dengan jumlah generasi maksimum sebesar 50.000 generasi, sedangkan pada kasus uji 2 didapatkan perbedaan nilai *fitness* terbaik sebesar 0,77%. Parameter ukuran populasi dan jumlah generasi memberikan pengaruh besar terhadap waktu eksekusi. Semakin tinggi nilai parameternya, proses eksekusi akan berlangsung lebih lama.

Kata kunci : *Library*, pustaka, *object-oriented design*, *object-oriented programming*, algoritma genetika

BAB I
PENDAHULUAN

1.1 Latar Belakang

Algoritma genetika merupakan pendekatan komputasional untuk menyelesaikan suatu masalah dengan memodelkan masalah tersebut menjadi proses biologi dari evolusi. Salah satu aplikasi algoritma genetika adalah pada permasalahan optimasi kombinasi, yaitu memperoleh suatu nilai solusi optimal terhadap suatu permasalahan yang memiliki banyak kemungkinan solusi. Secara garis besar tahapan dalam algoritma genetika ini dimulai dengan menetapkan suatu set solusi yang potensial dan melakukan perubahan dengan beberapa iterasi untuk mencapai solusi terbaik. Set solusi potensial ini ditetapkan di awal yang disebut dengan kromosom. Kromosom tersebut dibentuk secara acak, dapat berupa susunan angka biner yang dibangkitkan lalu dipilih. Keseluruhan set dari kromosom mewakili suatu populasi. Selanjutnya, kromosom-kromosom ini akan berevolusi dalam beberapa kali iterasi yang disebut dengan generasi. Generasi baru (*offspring*) dibangkitkan melalui proses pindah-silang (*crossover*) dan mutasi. Kromosom-kromosom *offspring* ini berevolusi dengan suatu kriteria kesesuaian (*fitness*) yang telah ditetapkan dimana hasil terbaik akan dipilih sedangkan yang lainnya diabaikan.

Library algoritma genetika merupakan perangkat lunak yang menyediakan lingkungan berdasarkan desain algoritma genetika yaitu representasi kromosom, fungsi evaluasi, operasi-operasi genetika (*crossover* dan mutasi) dan seleksi sehingga dapat digunakan untuk membantu penyelesaian masalah yang dapat diselesaikan dengan algoritma genetika. Desain *library* ini memungkinkan pengguna melakukan kustomisasi terhadap fitur yang telah disediakan.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dipaparkan di atas, maka rumusan masalah ditekankan pada:

1. Bagaimana merancang serta membangun perangkat lunak berupa *library* yang mengimplementasikan algoritma genetika dengan fungsi

evaluasi (fungsi *fitness*) dan parameter-parameter diberikan oleh *user library* yang menggunakannya.

2. Bagaimana merancang *library* algoritma genetika menggunakan metode *object-oriented design.*
3. Bagaimana mengimplementasikan *library* algoritma genetika menggunakan *object-oriented programming* dengan bahasa pemrograman C#.
4. Bagaimana menguji *library* algoritma genetika dengan kasus tertentu pada program aplikasi *user library*.

1.3 Batasan Masalah

Beberapa hal yang menjadi batasan masalah dalam penulisan tugas akhir ini antara lain:

1. *Library* algoritma genetika dibuat dalam bentuk *File Dynamic Link Library (DLL)* yang dirancang dengan *object-oriented programming.*
2. Algoritma yang digunakan *library* yaitu *simple genetic algorithm* (simple GA).
3. Bahasa pemrograman yang digunakan adalah C# dalam lingkungan .NET Framework 4.0.
4. Operasi genetika yang digunakan adalah *multi-point crossover*, mutasi *random uniform*, dan seleksi *roulette-wheel*. Pengkodean yang digunakan yaitu pengkodean bilangan riil (*real-value encoding*).
5. Pengujian *library* ini menggunakan kasus uji validasi yaitu membandingkan hasil perhitungan library algoritma genetika dengan hasil perhitungan referensi lain yang menggunakan algoritma genetika dan metode konvensional pada kasus uji tertentu.

1.4 Tujuan

Tujuan dari penyusunan tugas akhir (skripsi) ini yaitu menghasilkan suatu *library* algoritma genetika.

1.5 Manfaat

Manfaat yang diharapkan dapat diperoleh yaitu:

a. Bagi penyusun

1. Meningkatkan pemahaman penggunaan algoritma genetika.
2. Mampu mengimplementasikan algoritma genetika dalam *library* perangkat lunak.
3. Mampu membangun suatu *library* perangkat lunak yang dibutuhkan untuk menyelesaikan permasalahan yang ada.

b. Bagi pengguna

Untuk pengguna, bermanfaat mempermudah implementasi algoritma genetika dalam suatu pemrograman.

1.6 Sistematika Penulisan

Sistematika penulisan laporan skripsi ini adalah sebagai berikut:

BAB I Pendahuluan

Menjelaskan latar belakang, rumusan masalah, ruang lingkup, tujuan dan sistematika penulisan skripsi.

BAB II Dasar Teori

Membahas teori dasar dan teori penunjang yang digunakan.

BAB III Metode Penelitian

Membahas metode yang digunakan dalam penulisan yang terdiri dari studi literatur, perancangan perangkat lunak, implementasi perangkat lunak, pengujian dan analisis serta pengambilan kesimpulan dan saran.

BAB IV Perancangan dan Implementasi

Dalam bab ini dijelaskan langkah-langkah perancangan dan implementasi *library* algoritma genetika.

BAB V Pengujian

Menjelaskan langkah-langkah pengujian dari sistem yang telah dibuat dan membahas hasil pengujiannya.

BAB VI Penutup

Berisi kesimpulan yang diperoleh dari pembuatan dan pengujian *library*, serta saran–saran untuk pengembangan lebih lanjut.

BAB II
TINJAUAN PUSTAKA

2.1 Komputasi Evolusioner

Teori evolusi Charles Darwin pada 1859, memiliki esensi sebuah mekanisme optimisasi dan pencarian yang efektif pada alam. Prinsip Darwin tentang "*Survival of the fittest*" dapat digunakan sebagai titik awal untuk masuk dalam komputasi evolusioner. Makhluk hidup yang berevolusi menunjukkan perilaku kompleks yang optimal pada tiap-tiap tingkatan, mulai dari tingkat sel, organ, individu, hingga populasi.

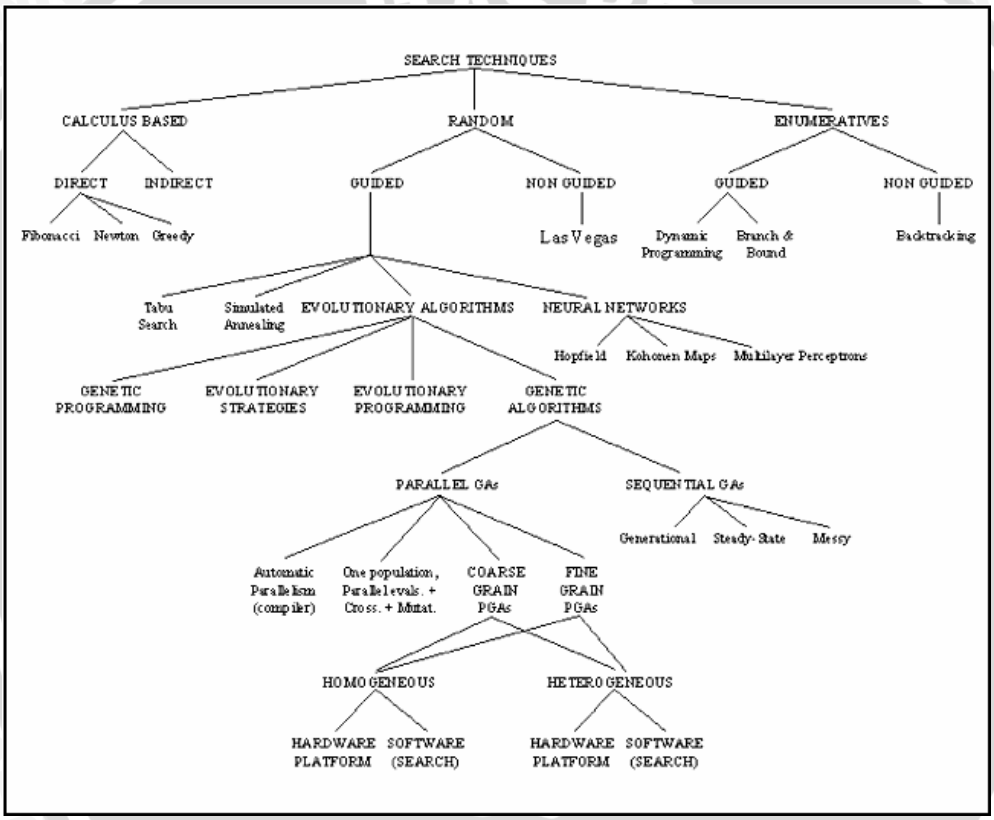
Spesies biologis telah memecahkan permasalahan yang berbasis kemungkinan atau acak yang terbukti dapat dibandingkan setara dengan metode klasik untuk proses optimisasi. Konsep evolusioner dapat diaplikasikan untuk masalah yang tidak memiliki solusi yang dapat diprediksikan, atau permasalahan yang kemungkinan memberikan hasil yang kurang memuaskan dengan metode konvensional. Sebagai hasilnya, algoritma evolusioner banyak digunakan untuk pendekatan praktis dari pemecahan suatu permasalahan.

2.1.1 Pengantar Komputasi Evolusioner

Teori seleksi alam mengemukakan bahwa semua tanaman dan hewan yang hidup saat ini adalah hasil dari proses adaptasi terhadap perubahan kondisi lingkungan selama jutaan tahun. Dalam satu rentang waktu yang sama, beberapa jenis organisme berbeda dapat hidup dan saling berkompetisi untuk mendapatkan sumber daya yang terdapat dalam ekosistem. Hanya organisme yang paling mampu mendapatkan sumber daya ini dan berhasil berkembang biak yang akan memiliki keturunan terbanyak pada generasi mendatang. Sedangkan yang kurang mampu beradaptasi cenderung kalah bersaing dan memiliki sedikit atau bahkan tidak memiliki keturunan di masa mendatang. Organisme yang pertama ini dapat disebut memiliki nilai *fitness* lebih tinggi daripada yang berikutnya. Dengan kata lain, organisme yang memiliki sifat-sifat yang dianggap baik ini yang akan memenuhi populasi dalam ekosistem.

Teknik komputasi evolusioner mengaplikasikan prinsip evolusi ini dalam sebuah algoritma yang dapat digunakan untuk mencari solusi optimal dari suatu

permasalahan. Aspek utama yang membedakan algoritma pencarian secara evolusioner dengan algoritma tradisional adalah basis kerja dari algoritma evolusioner yaitu pencarian berbasis populasi atau set nilai. Melalui keturunan dalam jumlah besar yang beradaptasi pada generasi selanjutnya, algoritma evolusioner melakukan sebuah proses pencarian secara langsung yang efisien. Secara umum pencarian secara evolusioner berjalan lebih baik daripada *random search* murni, dan tidak memiliki kelemahan mendasar dari teknik pencarian *local search* seperti *hill-climbing* yang berdasarkan gradien atau kemiringan.



Gambar 2.1 Teknik Pencarian [015]

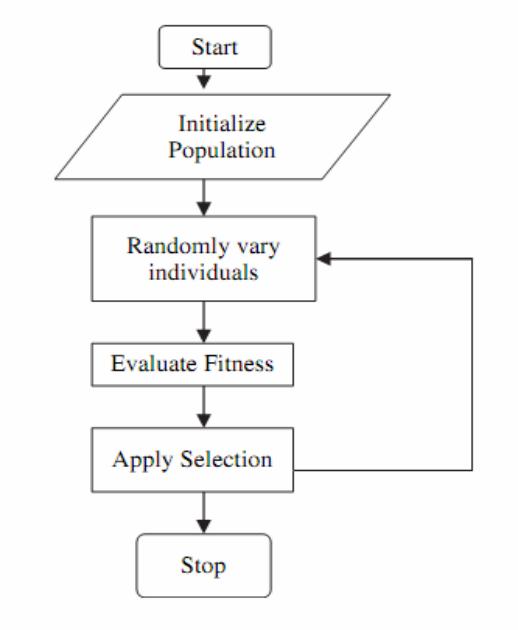
Dalam komputasi evolusioner, ada empat buah paradigma yang menjadi basis di lapangan, yaitu *genetic algorithms* (Holland, 1975), *genetic programming* (Koza, 1992, 1994), *evolutionary strategies* (Recheuberg, 1973), dan *evolutionary programming* (Forgel et al., 1966). Perbedaan yang mendasar dari paradigma ini terletak pada sifat dari skema pemodelan, operator reproduksi, dan metode seleksi. [015].

2.1.2 Keunggulan Komputasi Evolusioner

Komputasi evolusioner mendeskripsikan daerah pencarian dari semua algoritma evolusioner dan memberikan kelebihan praktis untuk beberapa permasalahan optimisasi. Beberapa keunggulan ini meliputi kesederhanaan konseptual dan kemampuan untuk menyelesaikan masalah yang tidak memiliki solusi. Pada bagian berikut ini akan dijelaskan sekilas tentang kelebihan ini dan menawarkan konsep perancangan algoritma evolusioner dalam penyelesaian permasalahan dunia nyata.

2.1.2.1 Kesederhanaan Konseptual

Keunggulan utama dari komputasi evolusioner adalah kesederhanaan konseptual Gambar 2.2 menunjukkan aplikasi dari algoritma evolusioner dalam fungsi optimisasi. Algoritma ini terdiri dari proses inisialisasi, variasi iterasi, dan seleksi berdasarkan indeks performa dari set solusi yang dihasilkan. Tidak diperlukan adanya informasi mengenai gradien di dalam algoritma ini. Melalui proses iterasi dari variasi yang acak dan proses seleksi, populasi dapat dibuat dibuat konvergen untuk menentukan solusi yang dianggap optimal. Efektifitas dari algoritma evolusioner tergantung dari variasi dan operator seleksi yang diaplikasikan pada representasi yang dipilih.



Gambar 2.2 Diagram Alir dari Algoritma Evolusioner

2.1.2.2 Menyelesaikan Masalah yang Tidak Memiliki Solusi

Keunggulan dari algoritma evolusioner meliputi kemampuan untuk melaukan referensi terhadap suatu permasalahan saat tidak ada kepakaran manusia dalam masalah tersebut. Walaupun kepakaran dari manusia seharusnya digunakan jika dibutuhkan dan tersedia, tetapi seringkali dapat dianggap kurang layak untuk keterlibatannya dalam proses penyelesaian masalah secara otomatis. Sistem kepakaran memiliki beberapa permasalahan sendiri, diantaranya jika pakar yang menjadi referensi tidak memiliki kualifikasi, tidak setuju dengan model, ataupun justru menyebabkan terjadinya error. Kecerdasan buatan bisa diaplikasikan pada beberapa permasalahan yang sukar yang membutuhkan kecepatan perhitungan sangat tinggi, tetapi tetap saja belum bisa bersaing dengan kecerdasan manusia. Fogel pada tahun 1995 menyatakan bahwa kecerdasan buatan mampu menyelesaikan masalah, tetapi mereka tidak menjawab permasalahan tentang bagaimana cara menyelesaikan masalah tersebut. Sebaliknya, komputasi evolusioner, menyediakan metode penyelesaian permasalahan tentang bagaimana cara menyelesaikan suatu masalah.

2.2 Algoritma Genetika

Algoritma Genetika adalah algoritma yang memanfaatkan proses seleksi alamiah yang dikenal dengan proses evolusi. Dalam proses evolusi, individu secara terus-menerus mengalami perubahan gen untuk menyesuaikan dengan lingkungan hidupnya. “Hanya individu-individu yang kuat yang mampu bertahan”. Proses seleksi alamiah ini melibatkan perubahan gen yang terjadi pada individu melalui proses perkembang-biakan. Dalam algortima genetika ini, proses perkembang-biakan ini menjadi proses dasar yang menjadi perhatian utama, dengar dasar berpikir: “Bagaimana mendapatkan keturunan yang lebih baik” [003]. Algoritma Genetika ini ditemukan oleh John Holland yang idenya tertulis dalam buku “Adaptation in natural and artificials systems” pada tahun 1975. Holland mengusulkan algoritma genetika sebagai metode heuristik berdasar pada “Kebertahanan pada *fittest* (*Survival of fittest*)” [015].

Prinsip algoritma genetika diambil dari teori Darwin yaitu setiap makhluk hidup akan menurunkan satu atau beberapa karakter ke keturunannya (anak).

Algoritma ini dimulai dengan kumpulan solusi yang disebut dengan populasi. Populasi disusun dari bermacam-macam individu. Setiap individu berisi *phenotype* (parameter), *genotype* (kromosom buatan atau *bit string*) dan *fitness* (fungsi objektif). Solusi-solusi dari sebuah populasi diambil dan digunakan untuk membentuk populasi yang baru. Hal ini dimotivasi dengan harapan bahwa populasi yang baru dibentuk tersebut akan lebih baik daripada yang lama. Dalam hal ini, individu dilambangkan dengan sebuah nilai *fitness* yang akan digunakan untuk mencari solusi terbaik dari persoalan yang ada.

Beberapa istilah penting dalam algoritma genetika [003], antara lain:

1) *Genotype* (Gen) adalah sebuah nilai yang menyatakan satuan dasar yang membentuk suatu arti tertentu dalam satu kesatuan gen yang dinamakan kromosom. Dalam algoritma genetika, gen ini bisa berupa nilai biner, *float*, *integer* maupun karakter, atau kombinatorial.
2) Allele, yaitu nilai dari gen.
3) Kromosom, merupakan gabungan gen-gen yang membentuk nilai tertentu.
4) Individu, menyatakan suatu nilai atau keadaaan yang menyatakan salah satu solusi yang mungkin dari permasalahan yang diangkat.
5) Populasi, merupakan sekumpulan individu yang akan diproses bersama dalam satu siklus evolusi.
6) Generasi, menyatakan satu-satuan siklus proses evolusi.
7) Nilai *fitness*, menyatakan seberapa baik nilai dari suatu individu atau solusi yang didapatkan.

2.2.1 Parameter-parameter Algoritma Genetika

2.2.1.1 Ukuran Populasi

Ukuran populasi menunjukkan seberapa banyak kromosom yang ada pada populasi (dalam satu generasi). Bila ada terlalu banyak kromosom, maka algoritma genetika memiliki beberapa kemungkinan untuk melakukan *crossover* dan hanya sebagian kecil *space* pencarian yang dieksplorasi. Dengan kata lain, jika ada terlalu banyak kromosom, algoritma genetika akan bergerak lambat. Penelitian menunjukkan setelah batas tertentu (terutama tergantung pada

pengkodean dan masalahnya) tidak berguna jika ukuran populasinya terus ditambah, karena tidak membuat masalah terselesaikan dengan cepat.

2.2.1.2 Jumlah Generasi

Generasi dapat dikatakan sebagai jumlah iterasi yang dilakukan terhadap proses evaluasi tiap-tiap populasi. Seperti halnya ukuran populasi, besarnya generasi akan mempengaruhi kecepatan konvergensi. Semakin besar jumlah generasi, mengakibatkan konvergensi yang lambat, tetapi bila jumlah generasi awal semakin kecil maka dapat terjadi konvergensi prematur. Untuk itu jumlah generasi yang tepat juga harus diperhitungkan dalam melakukan proses optimasi menggunakan algoritma genetika. Proses algoritma genetika akan dihentikan apabila jumlah generasi sudah terpenuhi.

2.2.1.3 Probabilitas *Crossover*

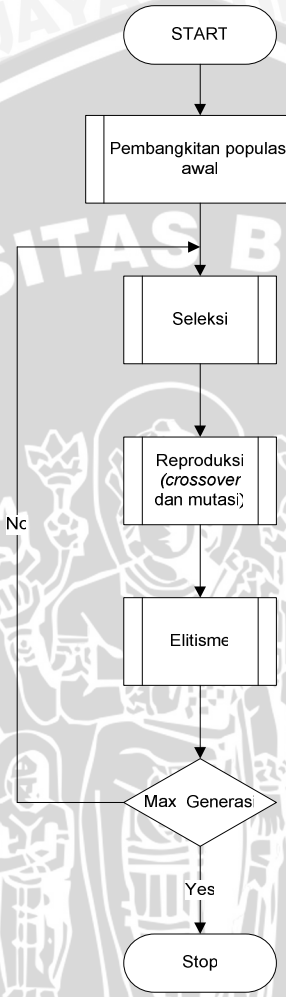
Probabilitas *Crossover* menunjukkan seberapa persen dari total kromosom yang akan melalui proses *crossover*. Bila tidak terjadi *crossover, offspring* (kromosom anak hasil *crossover*) merupakan salinan yang serupa dari induk (*parent*). Bila terjadi *crossover*, *offspring* disusun dari bagian-bagian kromosom induk (*parent*). Bila probabilitas *crossover* mencapai 100%, maka semua *offspring* disusun dari hasil *crossover*. Bila probabilitas *crossover* adalah 0%, maka keseluruhan generasi baru disusun dari salinan kromosom - kromosom populasi terdahulu yang serupa (namun hal ini tidak berarti generasi baru sama dengan generasi terdahulu). *Crossover* dilakukan dengan harapan kromosom - kromosom baru akan memiliki bagian – bagian yang baik dari kromosom – kromosom terdahulu dan kromosom – kromosom yang baru akan lebih baik daripada kromosom terdahulu.

2.2.1.4 Probabilitas Mutasi

Probabilitas mutasi menunjukkan seberapa sering bagian-bagian kromosom akan bermutasi. Bila tidak terjadi mutasi, *offspring* yang akan diambil adalah *offspring* setelah proses *crossover* (atau disalin) tanpa adanya perubahan. Bila mutasi terjadi, bagian-bagian kromosom berubah. Bila probabilitas mutasinya yaitu 100%, maka keseluruhan kromosom berubah, sedangkan bila probabilitas mutasinya yaitu 0%, maka kromosom tidak ada yang berubah.

2.2.2 Siklus Algoritma Genetika

Pada library algoritma genetika ini diimplementasi algoritma genetika simpel (*simpel genetic algorithm*). Gambar 2.3 di bawah ini menunjukkan siklus tersebut.



Gambar 2.3 Siklus Algoritma Genetika

Pada awal tahapan, algoritma akan melakukan pembangkitan sekumpulan alternatif solusi yang disebut juga dengan populasi. Populasi awal terdiri dari kromosom-kromosom. Kromosom-kromosom tersebut terdiri dari gen-gen dimana gen tersebut merupakan *encoding* yang digunakan dalam merepresentasikan masalah.

Kromosom-kromosom tersebut selanjutnya dievaluasi berdasarkan fungsi *fitness* yang telah ditentukan. Hasil dari evaluasi tersebut merupakan nilai *fitness*

dari kromosom. Tahapan yang selanjutnya akan dilakukan yaitu seleksi. Dalam seleksi akan dipilih individu-individu terbaik. Pada metode ini, setiap kromosom akan dibandingkan dengan bilangan acak yang dibangkitkan sebanyak jumlah kromosom. Setelah dibandingkan maka akan terbentuk susunan kromosom baru.

Proses selanjutnya adalah *crossover*. Tahapan ini dilakukan agar kromosom yang dianggap tidak pantas untuk dipertahankan pada generasi selanjutnya dapat digantikan dengan kromosom yang lebih baik dari induknya. Banyaknya kromosom yang akan mengalami *crossover* sesuai dengan probabilitas *crossover* yang telah ditentukan terlebih dahulu. Proses *crossover* akan menghasilkan individu baru yang disebut juga dengan *offspring*.

Pada individu baru (*offspring*) yang telah terbentuk diterapkan mutasi dengan tujuan untuk menghindari konvergen prematur. Banyaknya kromosom yang akan mengalami mutasi sesuai dengan probabilitas mutasi yang telah ditentukan nilainya. Individu-individu tersebut di evaluasi kembali berdasarkan fungsi *fitness*nya. Selanjutnya diimplementasikan elitisme. Elitisme dilakukan dengan tujuan menjaga agar individu bernilai *fitness* tertinggi tidak hilang selama proses evolusi dengan cara menyalin individu terbaik ke dalam generasi baru. Proses-proses mulai dari tahapan seleksi hingga mutasi yang akan menghasilkan generasi baru dilakukan berulang kali hingga tercapai maksimum generasi.

2.2.3 Komponen-komponen Utama Algoritma Genetika

Ada 6 komponen utama dalam algoritma genetika, yaitu pengkodean (*encoding*) kromosom, nilai *fitness*, pembangkitan populasi awal, seleksi, pindah-silang (*crossover*) dan mutasi.

2.2.3.1 Pengkodean Kromosom

Pengkodean merupakan proses merepresentasikan gen-gen individu. Proses tersebut dapat dilakukan menggunakan bit, bilangan real, pohon (*tree*), larik (*array*), *list* atau objek yang lainnya [015]. Pengkodean yang tepat sangat menentukan berhasil atau tidaknya proses algoritma genetika dalam menyelesaikan suatu permasalahan. Pengkodean yang tepat juga akan menentukan tingkat efisiensi komputasi yang digunakan.

2.2.3.2 Pembangkitan Populasi Awal

Membangkitkan populasi awal adalah proses membangkitkan sejumlah individu secara acak atau melalui prosedur tertentu. Syarat-syarat yang harus dipenuhi untuk menunjukkan suatu solusi harus benar-benar diperhatikan dalam pembangkitan setiap individunya [003]. Ukuran untuk populasi tergantung pada masalah yang akan diselesaikan dan jenis operator genetika yang akan diimplementasikan. Setelah ukuran populasi ditentukan, selanjutnya dilakukan pembangkitan tiap individunya.

Ada beberapa teknik dalam pembangkitan populasi awal yaitu random generator, pendekatan tertentu memasukkan nilai tertentu ke dalam gen dan permutasi gen.

1) Random Generator

   Inti dari cara ini adalah melibatkan pembangkitan bilangan random untuk nilai setiap gen sesuai dengan representasi kromosom yang digunakan. Bila menggunakan representasi biner, salah satu contoh penggunaan random generator adalah penggunaan rumus berikut untuk pembangkitan populasi awal:

   $$IPOP = \mathbf{round}\{\mathbf{random}(N_{ipops}, N_{bits})\}$$

   dengan IPOP adalah gen yang nantinya berisi pembulatan dari bilangan random yang dibangkitkan sebanyak N_{ipop} (Jumlah populasi) X N_{bits} (Jumlah gen dalam tiap kromosom).

2) Pendekatan Tertentu (Memasukkan Nilai Tertentu ke dalam Gen)

   Cara ini adalah dengan memasukkan nilai tertentu ke dalam gen dari populasi awal yang dibentuk.

3) Permutasi Gen

   Salah satu cara permutasi gen dalam pembangkitan populasi awal adalah penggunaan permutasi Josephus dalam kasus TSP.

2.2.3.3 Nilai *fitness*

Nilai *fitness* adalah nilai yang menyatakan baik tidaknya suatu solusi (individu). Nilai *fitness* ini yang dijadikan acuan dalam mencapai nilai optimal dalam algoritma genetika. Algoritma genetika bertujuan mencari individu dengan

nilai *fitness* yang paling tinggi [003]. Nilai *fitness* suatu kromosom dapat dihitung dengan menggunakan fungsi objektif.

Dalam TSP, karena TSP bertujuan meminimalkan jarak, maka nilai *fitness*nya adalah inversi dari total jarak dari jalur yang didapatkan [003]. Cara melakukan inversi bisa menggunakan rumus 1/*x* atau 100000-*x*, dengan *x* adalah total jarak dari jalur yang didapatkan.

2.2.3.4 Seleksi

Setiap kromosom yang terdapat dalam populasi akan melalui proses seleksi untuk dipilih menjadi orangtua. Seleksi dilakukan untuk mendapatkan calon induk yang baik [003]. Sesuai dengan teori Evolusi Darwin maka kromosom yang baik akan bertahan dan menghasilkan keturunan yang baru untuk generasi selanjutnya. Langkah pertama yang dilakukan dalam seleksi ini adalah pencarian nilai fitness. Semakin tinggi nilai fitness suatu individu, semakin besar kemungkinannya untuk terpilih [003].

Terdapat beberapa metode yang digunakan pada proses seleksi, antara lain *Rank Selection, Roulette Wheel Selection,* dan *Tournament Selection*.

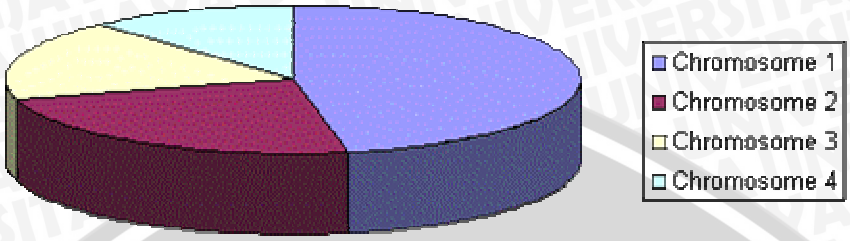
2.2.3.4.1 *Rank Selection*

Pada pengkodean ranking, kromosom pada populasi diranking sesuai dengan nilai *fitness*-nya, kemudian kromosom diberi nilai *fitness* yang baru sesuai dengan rankingnya. Kromosom dengan ranking terbawah akan mendapat nilai *fitness* 1, rangking terbawah kedua mendapat nilai *fitness* 2, demikian seterusnya. Kromosom dengan ranking terbaik akan mendapat nilai *fitness* N [015].

2.2.3.4.2 *Roulette Wheel Selection*

Pada *Roulette Wheel Selection,* kromosom akan dipilih secara acak ditentukan dengan memperhitungkan nilai *fitness* masing-masing kromosom. Semakin besar nilai *fitness* suatu kromosom, semakin besar pula peluang kromosom tersebut untuk terpilih sebagai *parent* (kromosom induk). Pengkodean *roulette wheel* dapat dianalogikan seperti permainan roda putar (*roulette wheel*). Pada permainan roda putar, lingkaran roda dibagi menjadi beberapa wilayah. Pada *roulette wheel selection,* lebar suatu wilayah kromosom ditentukan menurut nilai

fitness-nya, semakin besar nilai *fitness*-nya maka akan semakin besar wilayahnya, dan semakin besar pula peluang kromosom tersebut untuk terpilih [015].



Gambar 2.4 *Roulette Wheel Selection*

Roulette Wheel Selection dapat disimulasikan dengan algoritma sebagai berikut [015]:

1) **[Sum]** Jumlahkan semua nilai *fitness* tiap-tiap kromosom pada populasi S.
2) **[Select]** *Generate* bilangan *random* pada interval (0,S)-r.
3) **[Loop]** secara sekuensial dari kromosom pertama, jumlahkan nilai *fitness* kromosom-s. apabila pada kromosom ke-*i* s>r maka berhenti, maka kromosom *i* terpilih sebagai kandidat *parent.*

2.2.3.4.3 *Tournament Selection*

Pada metode *Tournament Selection*, ditetapkan suatu nilai *tour* untuk individu-individu yang dipilih secara random dari suatu populasi. Individu-individu yang terbaik dalam kelompok ini akan diseleksi sebagai induk. Parameter yang digunakan pada metode ini adalah ukuran *tour* yang bernilai antara 2 sampai N (jumlah individu dalam suatu populasi).

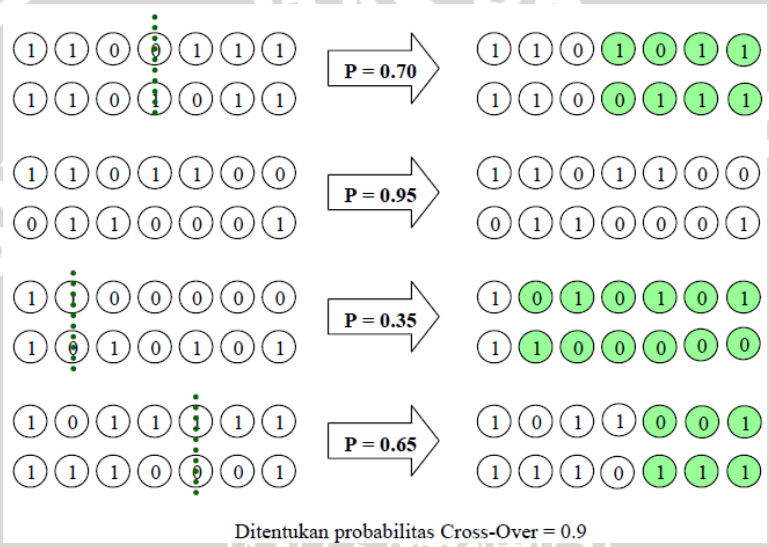
2.2.3.5 *Crossover* (Pindah-silang)

Crossover merupakan salah satu operator dalam algoritma genetika yang melibatkan dua induk untuk menghasilkan keturunan yang baru. *Crossover* dilakukan dengan melakukan pertukaran gen dari dua induk secara acak. Proses *Crossover* dilakukan pada setiap individu dengan probabilitas *Crossover* yang telah ditentukan [003]. Operasi ini tidak selalu dilakukan pada semua individu yang ada. Bila *crossover* tidak dilakukan, maka nilai dari induk akan diturunkan

kepada keturunannya. Probabilitas keberhasilan operasi *crossover* dinyatakan dengan P_c .

2.2.3.5.1 *Crossover* Satu Titik

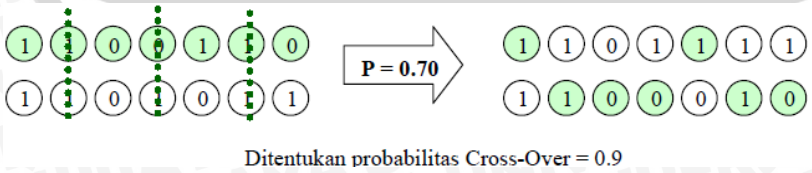
Crossover satu titik (*Single Point Crossover*) dan banyak titik biasanya digunakan untuk representasi kromosom dalam biner. Pada *crossover* satu titik, posisi *crossover* k (k=1,2,...,N-1) dengan N=panjang kromosom diseleksi secara acak. Variabel-variabel ditukar antar kromosom pada titik tersebut untuk menghasilkan anak.[001]



Gambar 2.5 Ilustrasi *Single Point Crossover* [001]

2.2.3.5.2 *Crossover* Banyak Titik

Pada *crossover* banyak titik, m posisi penyilangan k_i (k=1,2,...,N-1, i=1,2,...,m) dengan N= panjang kromosom diseleksi secara acak dan tidak diperbolehkan ada posisi yang sama serta diurutkan naik. Variabel-variabel ditukar antar kromosom pada titik tersebut untuk menghasilkan anak. Gambar 2.6 mengilustrasikan *crossover* lebih dari dua titik. [001]



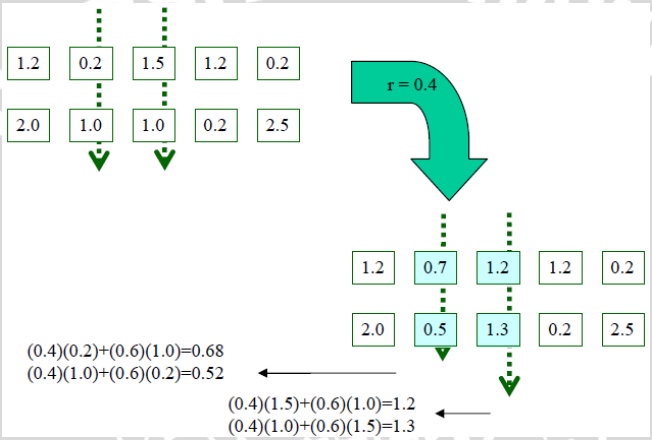
Gambar 2.6 Ilustrasi *Crossover* Lebih Dua Titik [001]

2.2.3.5.3 *Crossover* Aritmatika

Crossover aritmatika digunakan untuk representasi kromosom berupa bilangan *float* (pecahan). *Crossover* ini dilakukan dengan menentukan nilai r sebagai bilangan acak lebih dari 0 dan kurang dari 1. Selain itu juga ditentukan posisi dari gen yang dilakukan *crossover* menggunakan bilangan acak. Gambar 2.7 mengilustrasikan bagaimana *crossover* aritmatika bekerja. Nilai baru dari gen pada anak mengikuti rumus 2.1 dan 2.2. [001]

$x_1'(k) = r \,.\, x_1(k) + (1\text{-}r) \,.\, x_2(k)$ ..... (2.1)

$x_2'(k) = r \,.\, x_2(k) + (1\text{-}r) \,.\, x_1(k)$ ..... (2.2)



Gambar 2.7 *Crossover* Aritmatika [001]

2.2.3.6 Mutasi

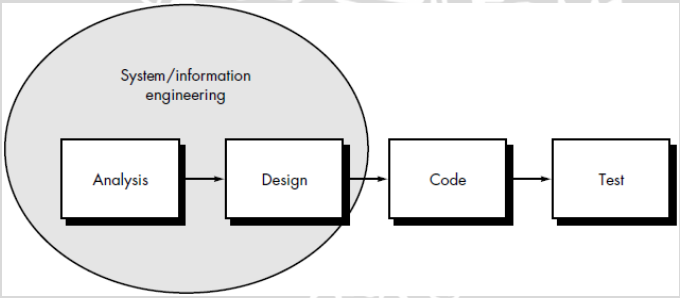
Setelah *crossover* dilakukan, proses reproduksi dilanjutkan dengan mutasi. Hal ini dilakukan untuk menghindari solusi-solusi dalam populasi memiliki nilai lokal optimum. Mutasi adalah proses mengubah gen dari keturunan secara random. Tidak setiap gen selalu dimutasi tetapi mutasi dikontrol dengan probabilitas tertentu yang disebut dengan *mutation rate* (probabilitas mutasi) dinotasikan dengan P_m. Pada pengkodean biner, mutasi mengubah bit 0 menjadi 1 dan sebaliknya bit 1 menjadi bit 0. Contoh mutasi pada pengkodean biner: 1**1**001001 => 1**0**001001.

2.3 Rekayasa Perangkat Lunak

Kompleksitas sistem yang semakin tinggi membuat beban pengembangan perangkat lunak semakin sulit [006]. Oleh karena itu diperlukan adanya metode

pengembangan untuk menghasilkan perangkat lunak yang berkualitas. Dalam hal ini dibutuhkan suatu rekayasa terhadap perangkat lunak (*Software Engineering*). Menurut IEEE, rekayasa perangkat lunak adalah penerapan yang sistematis, disiplin, serta pendekatan yang terukur terhadap pengembangan, pengoperasian, dan pemeliharaan perangkat lunak. [012].

Untuk menyelesaikan masalah dalam lingkungan industri, seorang *software engineer* atau sekumpulan *software engineer* harus menggabungkan strategi pengembangan yang meliputi proses, metode, dan alat bantu. Strategi ini yang kemudian sering disebut sebagai model proses (*process model*) atau paradigma rekayasa perangkat lunak (*software engineering paradigm*) . Model proses untuk rekayasa perangkat lunak dipilih sesuai dengan sifat dari proyek dan aplikasi yang akan dibuat. Salah satu dari model proses yang digunakan adalah *linear sequential model*. *Linear sequential model* biasa disebut sebagai *classic life cycle* atau *waterfall model*. Model proses *waterfall* ini merekomendasikan pendekatan yang sistematis dan terurut (*systematic and sequential approach*) untuk pengembangan perangkat lunak yang dimulai dari analisis kebutuhan (*requirement analysis*), perancangan (*design*), implementasi (*coding*), pengujian (*testing*), dan pemeliharaan (*maintenance*) [012].



Gambar 2.8 Linear Sequential Model [012]

Proses analisis kebutuhan, perancangan, implementasi, dan pengujian perangkat lunak dapat dikelompokkan menjadi dua metode, yaitu metode struktural dan metode berorientasi objek. Pada *library* algoritma genetika ini digunakan metode berorientasi objek yang menggunakan bahasa pemodelan UML (*Unified Modelling Language*). UML adalah bahasa untuk menggambarkan (*visualizing*), menspesifikasikan (*specifying*), membangun (*constructing*), dan mendokumentasikan (*documenting*) artefak dari sebuah sistem perangkat lunak

[008]. UML dirancang oleh Grady Booch, Ivar Jacobson, dan James Rumbaugh untuk menyatukan bermacam - macam bahasa pemodelan dan metode berorientasi objek dengan cara menggabungkan bahasa pemodelan dan metode berorientasi objek terbaik yang telah ada [008].

2.3.1 *Object-Oriented Design* (OOD)

Sistem berorientasi objek terdiri dari interaksi objek-objek yang mengolah *state* lokal objek itu sendiri dan menyediakan operasi-operasi pada *state* tersebut. Representasi *state* bersifat *private* dan tidak dapat diakses dari luar objek. Proses *object-oriented design* melibatkan proses desain kelas objek (*object class*) dan hubungan antar kelas-kelas tersebut. *OOD* merupakan bagian dari pengembangan *object-oriented*. *OOD* juga merupakan salah satu strategi pengembangan berorientasi objek pada proses perancangan, yaitu *object-oriented analysis, object-oriented design,* dan *object-oriented programming*.

Manfaat dari pendekatan *object-oriented design* ini yaitu bahwa sistem berorientasi objek lebih mudah diubah daripada sistem yang dikembangkan dengan pendekatan lainnya, karena objek bersifat independen. Objek-objek dapat dipahami dan dimodifikasi sebagai entitas yang berdiri sendiri (*stand-alone*). Mengubah implementasi sebuah objek ataupun menambah layanannya tidak akan mempengaruhi objek sistem lainnya.

Objek merupakan komponen yang bersifat *reusable* (dapat dipakai ulang) karena merupakan enkapsulasi dan operasi-operasi yang independen. Desain dapat dikembangkan dengan menggunakan objek-objek yang telah dibuat pada desain sebelumnya. Hal ini dapat mengurangi biaya desain, *programming* dan validasi. Hal ini juga dapat meyebabkan penggunaan objek standar (karenanya akan memperbaiki *design understandability*) dan mengurangi resiko dalam pengembangan perangkat lunak.

Beberapa metodologi *object-oriented design* yang telah dikenal yaitu Coad and Yourdon, 1990; Robinson, 1992; Jacobson, dkk, 1993; Graham, 1994; Booch, 1994; UML (*Unified Modelling Language*) oleh Rumbaugh, dkk, 1999. Dalam perancangan *library* algoritma genetika ini yang digunakan yaitu UML.

2.3.1.1 Proses *Object-Oriented Design*

Secara umum, proses *OOD* memiliki beberapa tahapan, antara lain [016]:

1) Memahami dan mendefinisikan konteks dan cara pemakaian sistem.
2) Merancang arsitektur sistem.
3) Mengidentifikasi objek-objek pokok dalam sistem.
4) Mengembangkan model desain.
5) Menentukan antarmuka objek.

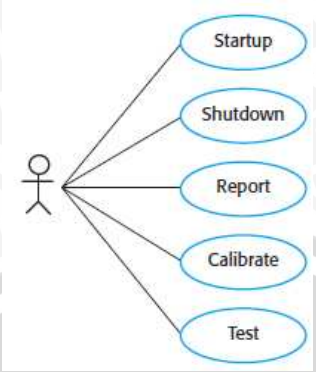
2.3.1.1.1 Konteks Sistem dan Model Pemakaian Sistem

Tahap pertama dalam setiap perancangan perangkat lunak adalah mengembangkan pemahaman mengenai *relationship* (hubungan) antara perangkat lunak yang sedang dirancang dengan lingkungan luar. Dengan memahami hal ini, akan membantu memutuskan bagaimana menyediakan sistem yang dibutuhkan dan bagaimana struktur sistem dapat berkomunikasi dengan lingkungannya.

Konteks sistem dan model pemakaian sistem merepresentasikan hubungan dua model yang saling melengkapi antara sistem dengan lingkungannya.

1) Konteks sistem merupakan model statis yang mendeskripsikan sistem lain yang ada dalam lingkungan sistem.
2) Model pemakaian sistem merupakan model dinamis yang mendeskripsikan cara sistem berinteraksi dengan lingkungannya.

Model konteks sistem dapat direpresentasikan menggunakan asosiasi dimana dibuat blok diagram sederhana dari arsitektur sistem secara keseluruhan. Kemudian dikembangkan dengan membagi-bagi model subsistem menggunakan pemodelan UML. Dengan kata lain model konteks sistem menggunakan model subsistem untuk menggambarkan sistem yang lain. Ketika memodelkan interaksi sistem dengan lingkungannya dapat digunakan pendekatan abstrak yang tidak terlalu rinci. Pendekatan ini mengembangkan model *use-case* dimana setiap *use-case* mewakili interaksi dengan sistem. Pada model *use-case*, setiap interaksi diberi nama dalam elips dan entitas luar yang terlibat dalam interaksi ditunjukkan dengan hubungan garis.



Gambar 2.9 *Use-case model*

2.3.1.1.2 Desain Arsitektur Sistem

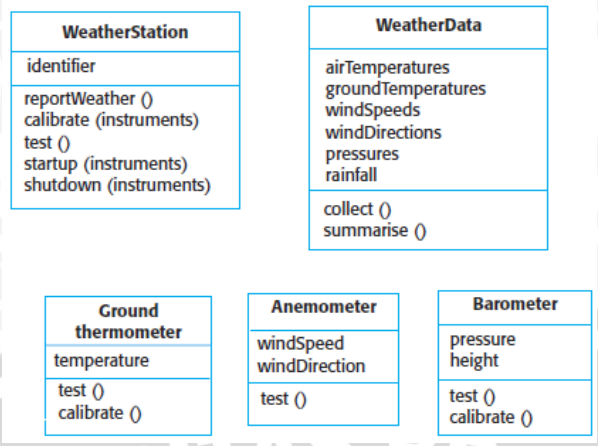
Setelah interaksi sistem perangkat lunak dirancang dan lingkungan sistem telah didefinisikan, langkah selanjutnya yaitu mendesain arsitektur sistem dengan menggunakan informasi yang telah diperoleh pada tahap sebelumnya.

2.3.1.1.3 Identifikasi Objek

Mengidentifikasi objek atau kelas objek merupakan bagian tersulit dalam *object-oriented design*. Tidak ada 'rumus ajaib' dalam identifikasi objek. Hal ini bergantung pada kemampuan, pengalaman dan domain pengetahuan dari perancang sistem.

Ada beberapa usulan cara mengidentifikasi kelas objek:

1) Gunakan analisis gramatikal deskripsi bahasa alami suatu sistem. Objek dan atribut dianalogikan dengan kata benda, operasi-operasi atau layanan sebagai kata kerja (Abbott, 1983).
2) Gunakan entitas nyata dalam domain aplikasi, misalnya pesawat, aturan (manajer), interaksi (pertemuan), lokasi (kantor), unit organisasi (perusahaan) dan sebagainya (Shlaer dan Mellor, 1988; Coad dan Yourdon, 1990; Wirfs-Brock, dkk, 1990).
3) Gunakan pendekatan perilaku dimana perancang terlebih dulu harus memahami perilaku sistem secara keseluruhan (Rubin dan Goldberg, 1992).
4) Gunakan analisis berbasis skenario dimana berbagai skenario pemakaian sistem diidentifikasi dan dianalisis (Beck dan Cunningham, 1989).



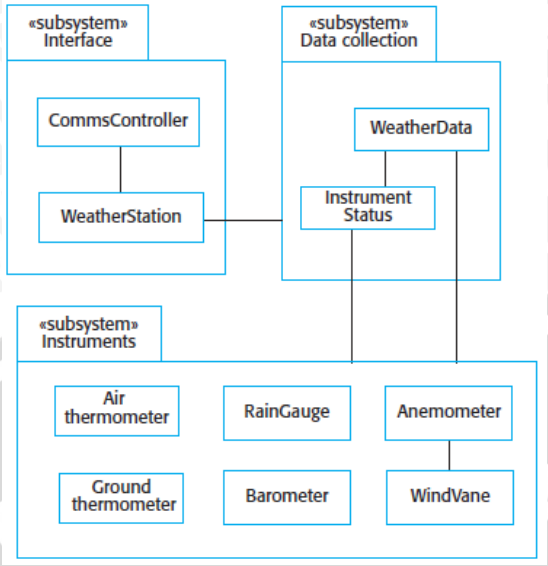
Gambar 2.10 Contoh Kelas Objek pada Sistem Stasiun Pemantau Cuaca

2.3.1.1.4 Model Desain

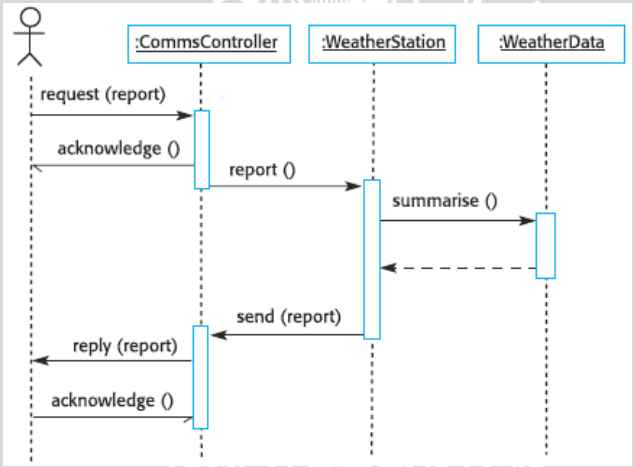
Model desain menggambarkan objek dan kelas objek dalam sistem dan hubungan antara entitas-entitas yang ada. Ada dua macam model desain yang seharusnya dirancang untuk mendeskripiskan *object-oriented design*:

1) Model statis (*static models*) mendeskripsikan struktur statis sistem menggunakan kelas objek dan hubungannya (*relationship*). Hubungan-hubungan penting yang harus didokumentasikan pada tahap ini yaitu *generalisation relationships, use/used-by relationships,* dan *composition relationships.*

2) Model dinamis (*dynamic models*) mendeskripsikan struktur dinamis sistem dan menunjukkan interaksi antara objek-objek dalam sistem (bukan kelas objek). Interaksi yang harus didokumentasikan meliputi urutan dari permintaan yang dibuat oleh objek dan langkah-langkah interaksi objek sesuai aturan sistem.



Gambar 2.11 Contoh *Subsystem Models* Model statis pada sistem stasiun pemantau cuaca



Gambar 2.12 Contoh *Sequence Models* Model dinamis pada sistem stasiun pemantau cuaca

2.3.1.1.4 Spesifikasi Antarmuka objek (*Object Interface*)

Objek antarmuka harus ditentukan sehingga objek dan komponen lainnya dapat dirancang paralel. Desainer harus menghindari merancang representasi antarmuka namun harus menyembunyikan hal ini dalam objek itu sendiri. UML menggunakan *class diagram* sebagai spesifikasi antarmuka.

```
interface WeatherStation {

   public void WeatherStation () ;

   public void startup () ;
   public void startup (Instrument i) ;

   public void shutdown () ;
   public void shutdown (Instrument i) ;

   public void reportWeather ( ) ;

   public void test () ;
   public void test ( Instrument i ) ;

   public void calibrate ( Instrument i) ;

   public int getID () ;

} //WeatherStation
```

Gambar 2.13 Antarmuka sistem stasiun pemantau cuaca dalam *Java*

2.3.2 Implementasi

Implementasi perangkat lunak dilakukan untuk merealisasikan desain perangkat lunak. *Library* algoritma genetika pada skripsi ini menggunakan bahasa pemrograman berorientasi objek. Bahasa pemrograman berorientasi objek yang digunakan adalah C#. Bahasa C# memiliki fitur menarik [017]:

- Bahasa pemrograman Modern

  C# merupakan versi modern dari C++. Selama ini kita memiliki bahasa C yang telah digunakan secara luas. C++ diciptakan untuk menambah fitur berorientasi objek pada bahasa C dan C++ telah menjadi bahasa pemrograman yang membangun aplikasi "nyata" untuk Windows. C++ telah digunakan untuk menuliskan infrastruktur dan aplikasi level rendah, sedangkan pengembang Visual Basic (VB) menuliskan aplikasi bisnis. C# menghadirkan paradigma pengembangan yang cepat dari VB menuju dunia bahasa C++, dengan beberapa perubahan yang luar biasa. C# mengambil keuntungan dari .NET Framework yang artinya kita memiliki akses ke mesin yang *powerful*, seperti yang dialami VB selama beberapa tahun.

- *Type-Safe*

  Bahasa C# merupakan *type-safe* yang berarti beberapa hal. Sebagai contoh, kita tidak bisa tidak melakukan inisialisasi variabel. Pada C++,

mudah sekali mendeklarasikan variabel dan menampilkan nilainya, apapun yang ada pada alamat memori variabel tersebut akan ditampilkan. Hal ini bisa mendatangkan malapetaka bagi aplikasi. *Compiler* C# akan memberitahu kita ketika kita mencoba memanipulasi variabel dimana nilainya belum diinisialisasi.

- Berorientasi Objek

  Ketika banyak yang beragumen bahwa C++ berorientasi objek, C# menuju ke level yang lain. Bahkan tipe data sederhana dapat diperlakukan sebagai objek, berarti *int* memiliki *method* yang terkait dengannya. Misalnya, kita dapat menggunakan *method* `ToString` untuk memperoleh nilai string dari suatu integer: `int Counter = 14; Console.Write(Counter.ToString());`

- Memiliki Sintaks yang Sederhana

  Meskipun C++ bahasa yang sangat *powerful,* namun bisa dikatakan tidak dianggap mudah. C# mencoba untuk menyederhanakan sintaks dengan lebih konsisten dan lebih logis juga menghilangkan beberapa fitur kompleks dari C++. Sebagai contoh, C# menjauhi dari *pointer*. Sebagai bahasa pemrograman yang *type-safe*, C# tidak mengijinkan manipulasi memori secara langsung, jadi *pointer* tidak lagi dibutuhkan pada C#.

- *Cross-language Capabilities*

  C# memiliki kemampuan untuk mengijinkan kita untuk *interoperate* dengan bahasa pemrograman lain di dalam *platform* .NET. Telah banyak dijelaskan tentang bagaimana kita membuat komponen pada satu bahasa pemrograman dan menurunkannya pada bahasa pemrograman lain dimana hal ini sesuatu yang sulit menjadi mungkin.

- Tidak Hanya Microsoft

  C# bukan tentang Microsoft lagi sekarang. Microsoft merilis C# ke ECMA/European Computer Manufactures Association (Java tidak melakukannya). Sebagai tambahan, proyek Mono merupakan suatu usaha untuk membuat versi *open source* dari .NET Framework dan versi *open source* dari C# yang semuanya berjalan pada linux.

2.3.3 *Dynamic Link Library*

Library algoritma genetika pada skripsi ini berupa file *dynamic link library*. *Dynamic Link Library* (DLL) merupakan modul yang berisi fungsi dan data yang dapat digunakan oleh modul yang lain (aplikasi atau DLL).

DLL menyediakan suatu cara untuk memodularkan aplikasi sehingga fungsionalitas aplikasi dapat diperbarui dan digunakan ulang (*reused*) dengan mudah. DLL juga membantu mereduksi penggunaan memori ketika beberapa aplikasi menggunakan fungsionalitas pada saat yang bersamaan, karena walaupun setiap aplikasi mendapat salinan data DLL, aplikasi tersebut berbagi (*share*) kode DLL yang digunakan. *Dynamic linking* memiliki beberapa keunggulan dibandingkan dengan *static linking*, antara lain: proses-proses yang memanggil DLL yang sama pada basis alamat yang sama berbagi salinan tunggal DLL dalam *physical memory* sehingga dapat menghemat penggunaan memori; ketika fungsi-fungsi yang terdapat pada DLL berubah maka aplikasi yang menggunakannya tidak perlu dikompilasi ulang atau dilink ulang selama argumen fungsi, pemanggilan dan nilai kembalian tidak berubah; program yang ditulis pada bahasa pemrograman yang berbeda dapat memanggil fungsi DLL yang sama selama program tersebut mengikuti aturan pemanggilan fungsi tersebut.

BAB III
METODE PENELITIAN

Pada bab ini dipaparkan langkah-langkah yang akan dilakukan dalam perancangan, implementasi dan pengujian *library* algoritma genetika yang akan dibuat. Metode penelitian yang digunakan pada penyusunan skripsi ini adalah:

3.1 Studi Literatur

Studi literatur menjelaskan dasar teori yang digunakan untuk menunjang penulisan skripsi. Teori-teori pendukung tersebut meliputi:

a. Algoritma Genetika
   - Parameter-parameter Algoritma Genetika
   - Siklus Algoritma Genetika
   - Komponen-komponen Utama Algoritma Genetika (Pengkodean Kromosom, Pembangkitan Populasi Awal, Nilai *Fitness*, Seleksi, *Crossover*, Mutasi)

b. Rekayasa Perangkat Lunak
   - *Object-Oriented Design*
   - Implementasi
   - *Dynamic Link Library*

3.2 Analisis Kebutuhan

Analisis kebutuhan bertujuan untuk mendapatkan semua kebutuhan yang diperlukan dari sistem yang akan dibangun. Metode analisis yang akan digunakan adalah *Object-Oriented Analysis* yaitu menggunakan bahasa pemodelan UML (*Unified Modelling Language*). Diagram *Use Case* digunakan untuk mendeskripsikan kebutuhan-kebutuhan dan fungsionalitas sistem dari perspektif *end-user*. Analisis kebutuhan dilakukan dengan mengidentifikasi semua kebutuhan *library* algoritma genetika kemudian dimodelkan dalam diagram *use case*.

3.3 Perancangan

Perancangan *library* algoritma genetika dilakukan setelah semua kebutuhan sistem didapatkan melalui tahap analisis kebutuhan. Perancangan *library* algoritma genetika berdasarkan *Object-Oriented Analysis* dan *Object-Oriented Design* menggunakan bahasa pemodelan UML (*Unified Modelling Language*). Perancangan *library* algoritma genetika dilakukan dengan mengidentifikasi kelas-kelas dan *interface-interface* yang dibutuhkan yang dimodelkan dalam diagram kelas (*class diagram*). Relasi antar objek yang telah diidentifikasi dimodelkan dalam diagram sekuensial (*sequence diagram*). Diagram sekuensial menggambarkan interaksi antar objek yang disusun dalam urutan waktu.

3.4 Implementasi

Implementasi dilakukan dengan mengacu kepada perancangan *library* algoritma genetika. Implementasi *library* berupa *file dynamic link library* dilakukan dengan menggunakan bahasa pemrograman berorientasi objek yaitu menggunakan bahasa pemrograman C#.

3.5 Pengujian

Pengujian dilakukan untuk menunjukkan bahwa *library* yang telah dirancang bekerja sesuai dengan fungsi yang telah ditentukan dan benar perhitungannya. Pengujian yang dilakukan meliputi *validasi library* dan pengujian *execution time*. Validasi *library* dilakukan dengan membandingkan hasil perhitungan (nilai *fitness* terbaik) dengan hasil perhitungan referensi lain dan metode konvensional pada kasus uji tertentu. Pengujian *execution time* bertujuan untuk mengetahui waktu eksekusi yang dibutuhkan saat proses evolusi algoritma genetika pada *library* dijalankan.

3.6 Pengambilan Kesimpulan

Pada tahap ini, diambil kesimpulan dari hasil pengujian dan analisis terhadap *library* algoritma genetika yang telah dibangun. Tahap selanjutnya adalah pembuatan saran untuk perbaikan terhadap penelitian selanjutnya sehingga dapat menyempurnakan kekurangan-kekurangan yang ada.

BAB IV
PERANCANGAN DAN IMPLEMENTASI

Bab ini membahas tahapan-tahapan perancangan dan implementasi pada pengembangan *library* algoritma genetika dengan pendekatan *object-oriented design* dan *object-oriented programming*. Perancangan dilakukan dalam dua tahap. Tahap pertama yaitu analisis kebutuhan dan tahap kedua yaitu perancangan perangkat lunak. Pada tahap analisis kebutuhan dibuat daftar kebutuhan *user* dan pemodelan *use-case diagram* untuk menggambarkan kebutuhan *library*. Proses perancangan perangkat lunak terdiri dari dua tahap yaitu perancangan umum dan perancangan terinci. Perancangan umum meliputi perancangan arsitektur *library* berupa relasi antar *namespace* (paket) sebagai pemodelan secara keseluruhan. Perancangan terinci meliputi identifikasi objek dan *sequence diagram.* Pada identifikasi objek, objek-objek yang teridentifikasi selanjutnya digambarkan oleh *class diagram*, sedangkan *sequence diagram* menggambarkan desain model. Implementasi perangkat lunak menjelaskan lingkungan implementasi dan memaparkan detil kelas-kelas yang terdapat dalam *library* algoritma genetika.

4.1 Analisis Kebutuhan

Analisis kebutuhan bertujuan menggambarkan kebutuhan-kebutuhan yang harus disediakan oleh *library* sehingga dapat memenuhi kebutuhan pengguna sesuai tahapan-tahapan dalam algoritma genetika. Proses analisis kebutuhan mengambil acuan dari tahapan – tahapan dalam algoritma genetika dan hasil pengumpulan, pemahaman dan penetapan kebutuhan-kebutuhan yang ingin didapatkan pengguna. Analisis kebutuhan dimulai dengan identifikasi aktor yang terlibat dengan *library*, penjabaran daftar kebutuhan, kemudian dimodelkan ke dalam diagram *use-case*.

4.1.1 Identifikasi Aktor

Tahapan ini bertujuan untuk melakukan identifikasi terhadap aktor yang akan berinteraksi dengan *library* algoritma genetika. Tabel 4.1 memperlihatkan aktor beserta deskripsinya yang merupakan hasil dari proses identifikasi aktor.

Tabel 4.1 Deskripsi Aktor

Aktor	**Deskripsi Aktor**
User Library	*User Library* merupakan aktor pengguna yang akan mengoperasikan *library* algoritma genetika

4.1.2 Daftar Kebutuhan

Daftar kebutuhan yang dipaparkan pada Tabel 4.2 terdiri dari sebuah kolom yang menguraikan kebutuhan yang harus disediakan oleh *library* dan pada kolom lain ditunjukkan nama *use-case* yang akan menyediakan fungsionalitas masing-masing kebutuhan tersebut.

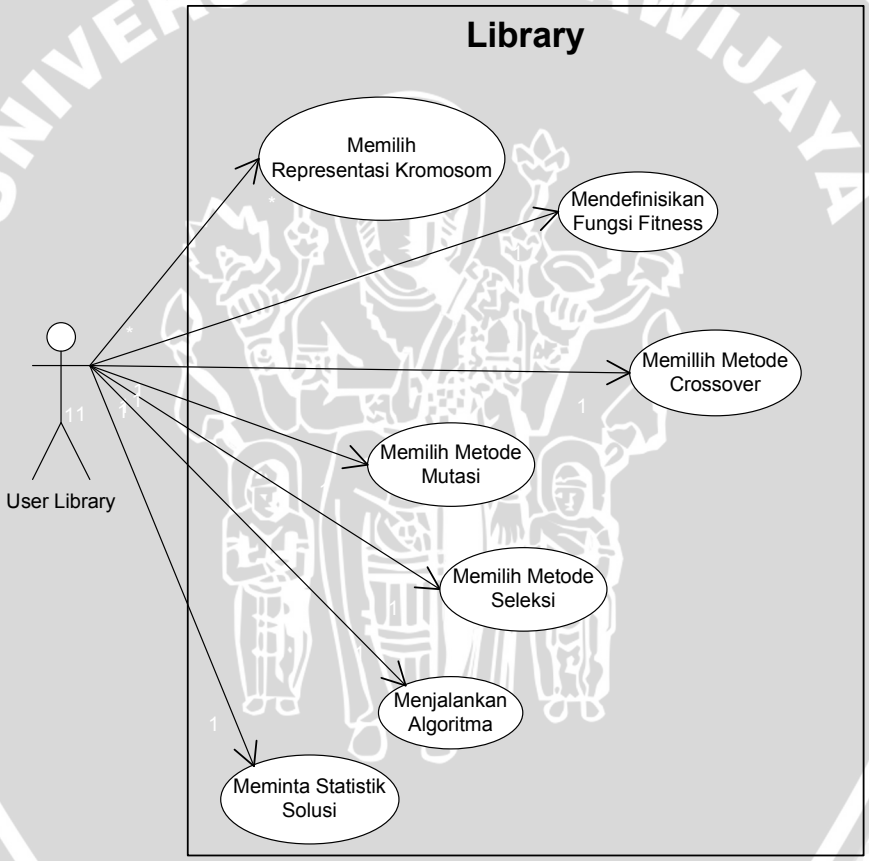
Tabel 4.2 Daftar Kebutuhan *Library* Algoritma Genetika

ID	***Requirements***	**Aktor**	**Nama *Use-Case***
F01	*Library* harus menyediakan representasi kromosom	*User Library*	Memilih representasi kromosom
F02	*Library* harus menyediakan *interface* operasi fitness	*User Library*	Mendefinisikan Fungsi Fitness
F03	*Library* harus menyediakan metode *crossover*	*User Library*	Memilih metode *crossover*
F04	*Library* harus menyediakan metode mutasi	*User Library*	Memilih metode mutasi
F05	*Library* harus menyediakan metode seleksi	*User Library*	Memilih metode seleksi
F06	*Library* harus menyediakan proses implementasi algoritma genetika (pencarian solusi)	*User Library*	Menjalankan algoritma
F07	*Library* harus menyediakan proses untuk menghitung dan menyimpan statistik populasi serta prosedur untuk mengembalikan statistik populasi yang diminta *user library*	*User Library*	Meminta statistik populasi

4.1.3 Diagram *Use-Case*

Tahap berikutnya yaitu memahami hubungan / relasi antara *library* dengan lingkungan eksternalnya. Hubungan ini dapat direpresentasikan dalam

Model Penggunaan Sistem. Model Penggunaan Sistem merupakan model dinamis yang mendeskripsikan interaksi *library* dengan lingkungannya. Model yang menggambarkan interaksi *library* dengan lingkungannya ini menggunakan pendekatan abstrak yang tidak melibatkan terlalu banyak detil. Pendekatan tersebut mengembangkan model *use-case* dimana setiap *use-case* menggambarkan interaksi dengan *library*. Kebutuhan-kebutuhan yang diperlukan oleh pengguna dan harus disediakan oleh *library* dimodelkan dengan model *use-case* (diagram *use-case*).



Gambar 4.1 Diagram use-case *library*

4.1.4 Deskripsi *use-case*

Setiap *use-case* tersebut dapat dideskripsikan dengan bahasa alamiah terstruktur. Hal ini membantu perancangan *library* terutama pada proses

identifikasi objek dan operasi-operasi yang akan dilakukan dalam *library*. Tabel 4.3 merupakan deskripsi *use-case* Memilih Representasi Kromosom.

Tabel 4.3 Deskripsi *use-case* Memilih Representasi Kromosom

Use-case	**Memilih Representasi Kromosom**
Aktor	***User Library***
Data	**Data yang diberikan *User Library* yaitu parameter kromosom, dan domain kromosom (*constraint*).**
Stimulus	**-**
Respon	**-**
Komentar	***User Library* memilih atau mendefinisikan representasi kromosom sesuai dengan yang dibutuhkannya yaitu pengkodean biner, real value, atau permutasional.**

Selanjutnya Tabel 4.4 menggambarkan deskripsi *use-case* Mendefinisikan Fungsi Fitness.

Tabel 4.4 Deskripsi *use-case* Mendefinisikan Fungsi Fitness

Use-case	**Mendefinisikan Fungsi Fitness**
Aktor	***User Library***
Data	**-**
Stimulus	**-**
Respon	**-**
Komentar	***User Library* mendefinisikan fungsi fitness. Fungsi Fitness harus didefinisikan sendiri oleh *User Library* kemudian *library* akan merujuk pada Fungsi Fitness yang telah didefinisikan *User Library*. Fungsi Fitness ini memberikan nilai kembalian berupa nilai fitness (*fitness value*).**

Kebutuhan selanjutnya yang harus disediakan *library* yaitu kebutuhan untuk Memilih Metode *Crossover*. Kebutuhan tersebut direpresentasikan oleh

use-case Memilih Metode *Crossover*. Tabel 4.5 merupakan deskripsi *use-case* Memilih Metode *Crossover*.

Tabel 4.5 Deskripsi *use-case* Memilih Metode *Crossover*

Use-case	**Memilih Metode *Crossover***
Aktor	***User Library***
Data	**-**
Stimulus	**-**
Respon	**-**
Komentar	***User Library* memilih metode *crossover* yang disediakan oleh *library*. *User Library* juga dapat mendefinisikan sendiri metode *crossover* di luar yang disediakan oleh *library*.**

Tabel 4.6 menunjukkan deskripsi *use-case* Memilih Metode Mutasi berdasarkan diagram *use-case* pada Gambar 4.1.

Tabel 4.6 Deskripsi *use-case* Memilih Metode Mutasi

Use-case	**Memilih Metode Mutasi**
Aktor	***User Library***
Data	**-**
Stimulus	**-**
Respon	**-**
Komentar	***User Library* memilih metode mutasi yang disediakan oleh *library*. *User Library* juga dapat mendefinisikan sendiri metode mutasi di luar yang disediakan oleh *library*.**

Tabel 4.7 menunjukkan deskripsi *use-case* Memilih Metode Seleksi berdasarkan diagram *use-case* pada Gambar 4.1.

Tabel 4.7 Deskripsi *use-case* Memilih Metode Seleksi

Use-case	**Memilih Metode Seleksi**
Aktor	***User Library***
Data	**-**
Stimulus	**-**
Respon	**-**
Komentar	***User Library* memilih metode seleksi yang disediakan oleh *library*. *User Library* juga dapat mendefinisikan sendiri metode seleksi di luar yang disediakan oleh *library*.**

Tabel 4.8 menunjukkan deskripsi *use-case* Menjalankan Algoritma berdasarkan diagram *use-case* pada Gambar 4.1.

Tabel 4.8 Deskripsi *use-case* Menjalankan Algoritma

Use-case	**Menjalankan Algoritma**
Aktor	***User Library***
Data	**Data yang diberikan *User Library* meliputi konfigurasi populasi dan maksimum generasi yang diperbolehkan. Populasi merupakan kumpulan-kumpulan set solusi (kromosom).**
Stimulus	**-**
Respon	**-**
Komentar	***User Library* menjalankan proses evolusi algoritma genetika untuk mencari solusi dari permasalahan yang telah didefinisikan sebelumnya.**

Tabel 4.9 menunjukkan deskripsi *use-case* Meminta Statistik Populasi berdasarkan diagram *use-case* pada Gambar 4.1.

Tabel 4.9 Deskripsi *use-case* Meminta Statistik Populasi

Use-case	**Meminta Statistik Populasi**
Aktor	***User Library***

Data	**Data yang diberikan *User Library* yaitu data populasi yang akan diobservasi statistiknya.**
Stimulus	***User Library request* statistik populasi yang ingin diobervasi antara lain: kromosom terbaik, fitness rata-rata, fitness maksimum atau fitness minimum.**
Respon	***User Library* menerima data statistik yang sesuai dengan *request*.**
Komentar	***User Library* meminta statistik populasi. *Request* dilakukan untuk memperoleh data kromosom terbaik, fitness rata-rata, fitness maksimum atau fitness minimum.**

4.2 Perancangan Perangkat Lunak

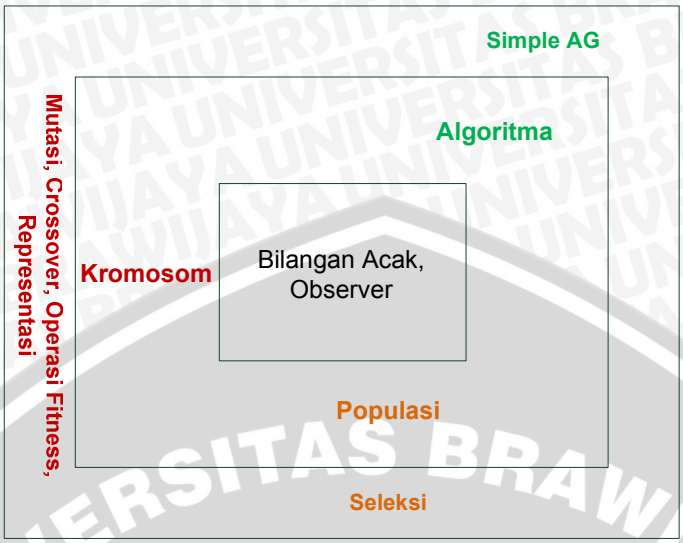
Perancangan perangkat lunak dilakukan dalam dua tahap yaitu perancangan umum dan perancangan terinci. Perancangan perangkat lunak pada skripsi ini menggunakan pendekatan desain berorientasi objek yang direpresentasikan dengan menggunakan UML (*Unified Modeling Language*). Pada proses desain dilakukan identifikasi terhadap kelas-kelas yang dibutuhkan yang dimodelkan dalam diagram kelas (*class diagram*), sedangkan hubungan interaksi antar objek yang telah diidentifikasi dimodelkan dalam diagram sekuensial (*sequence diagram*).

4.2.1 Perancangan Umum

Setelah interaksi antara *library* yang akan dirancang dengan lingkungan sistem telah ditentukan, informasi ini dapat digunakan sebagai dasar untuk merancang arsitektur *library*. Perancangan umum menggambarkan struktur *library* dan desain arsitektur *library* yang direpresentasikan berupa relasi antar *namespace*.

4.2.1.1 Struktur *Library* Algoritma Genetika

Struktur dasar *Library* Algoritma Genetika ditunjukkan pada diagram berikut ini (Gambar 4.2).



Gambar 4.2 Struktur *Library* Algoritma Genetika

Struktur *library* terdiri dari dua lapisan utama. Lapisan pertama terdiri dari unit-unit yang tidak berkaitan langsung dengan algoritma genetika namun implementasi unit-unit tersebut penting. *Library* algoritma genetika mengimplementasikan pembangkit (*generator*) bilangan acak juga menyediakan unit yang lebih spesifik terhadap algoritma genetika pada lapisan terendah yaitu unit untuk mengobservasi informasi statistik populasi. Fitur-fitur tersebut menyediakan fungsi-fungsi umum yang digunakan oleh unit-unit lainnya yang berada pada lapisan yang lebih tinggi dalam *library*. Unit-unit pada lapisan pertama ini dibagi menjadi beberapa kelompok unit.

Lapisan bagian tengah terdiri dari tiga unit seperti pada gambar 4.2. Fitur utama *library* diimplementasikan pada lapisan ini. Unit pertama yaitu unit kromosom. Unit kromosom beranggotakan unit-unit yang akan digunakan untuk merepresentasikan kromosom dan mendefinisikan perilakunya dalam sistem. Selain itu, pada unit ini juga terdapat unit-unit yang menangani operasi-operasi genetika yaitu *crossover*, mutasi, dan operasi *fitness*. Unit kedua yaitu unit populasi. Unit populasi merupakan unit yang mengatur sekumpulan kromosom (populasi). Di dalam unit populasi terdapat unit yang bertugas mengatur operasi seleksi. Unit yang terakhir yaitu unit algoritma. Unit algoritma mengimplementasikan proses evolusi algoritma genetika. Kedua lapisan ini menggambarkan inti dari *library* algoritma genetika.

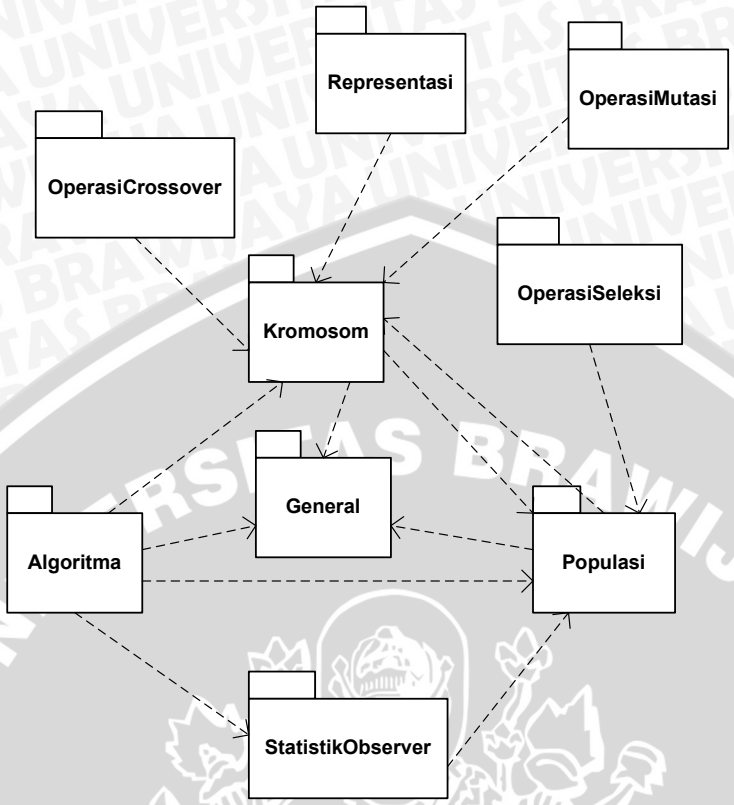
4.2.1.2 Desain Arsitektur *Library*

Unit-unit yang terdapat pada struktur *library* algoritma genetika, yang telah dipaparkan sebelumnya, nantinya akan beranggotakan kelompok kelas-kelas dan *interface*. Pengelompokan beberapa kelas dan *interface* dalam satu unit disebut juga paket (*package*) – untuk selanjutnya dalam skripsi ini disebut dengan *namespace*. Tabel 4.10 memaparkan daftar *namespace* yang menyusun *library* algoritma genetika.

Tabel 4.10 Daftar *namespace* Beserta Deskripsinya

Namespace	**Deskripsi**
Algoritma	Berisi kelas yang digunakan untuk mengimplementasikan algoritma genetika
Kromosom	Berisi *interface* dan kelas yang diperlukan untuk mengimplementasikan perilaku dan representasi kromosom serta operasi genetika
Kromosom.OperasiCrossover	Berisi implementasi operasi *crossover*
Kromosom.OperasiMutasi	Berisi implementasi operasi mutasi
Kromosom.Representasi	Berisi implementasi representasi kromosom
General	Berisi kelas umum yang digunakan oleh *library*
Populasi	Berisi kelas yang digunakan untuk implementasi populasi kromosom dan operasi genetika
Populasi.OperasiSeleksi	Berisi implementasi operasi seleksi
StatistikObserver	Berisi kelas yang digunakan untuk mengobservasi informasi statistik populasi

Desain arsitektur *library* algoritma genetika digambarkan berupa relasi antar *namespace* sebagai pemodelan sistem secara keseluruhan. Kelas-kelas yang merupakan anggota dari tiap-tiap *namespace* akan dipaparkan pada subbab Perancangan Terinci. Relasi antar *namespace* dapat dilihat pada gambar 4.3.



Gambar 4.3 Relasi antar *Namespace*

4.2.2 Perancangan Terinci

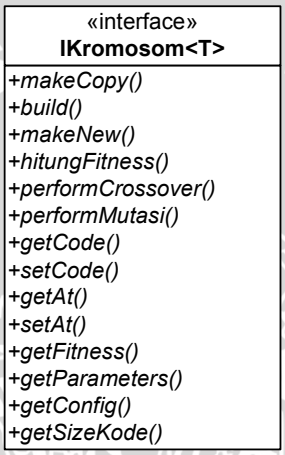
Perancangan terinci (detil) menggunakan diagram kelas dan diagram sekuensial sebagai pemodelan perangkat lunak. Perancangan terinci menjelaskan pola hubungan antar komponen-komponen detil (kelas dan objek) sehingga membentuk fungsi yang mampu melayani kebutuhan pengguna.

4.2.2.1 Identifikasi Objek

Pada tahap ini dilakukan identifikasi objek sesuai dengan sistem yang dirancang. Penentuan objek mengacu pada perancangan umum yang telah dilakukan sebelumnya. Objek-objek yang telah diidentifikasi merupakan representasi dari suatu kelas. Diagram kelas memberikan gambaran pemodelan elemen-elemen kelas yang membentuk sebuah sistem perangkat lunak. Elemen-elemen kelas meliputi atribut, *method* serta hubungan kelas (*relationship*) dengan kelas-kelas lain dalam sebuah sistem. Pada *library* algoritma genetika ini kelas-kelas dikelompokkan ke dalam beberapa *namespace* sesuai tabel 4.10.

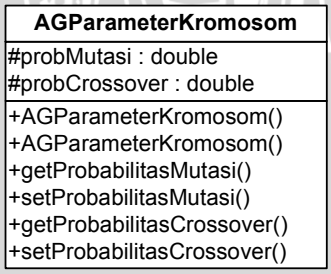
4.2.2.1.1 Kromosom

Kromosom merupakan objek utama pada algoritma genetika. Pada *library* ini, kromosom didefinisikan oleh IKromosom. IKromosom merupakan *interface* untuk implementasi aktual kromosom. IKromosom tidak mengimplementasikan *method* (perilaku) apapun. Implementasi *method* didefinisikan oleh kelas turunannya. Gambar 4.4 merupakan diagram kelas *interface* IKromosom.



Gambar 4.4 Diagram Kelas *interface* IKromosom

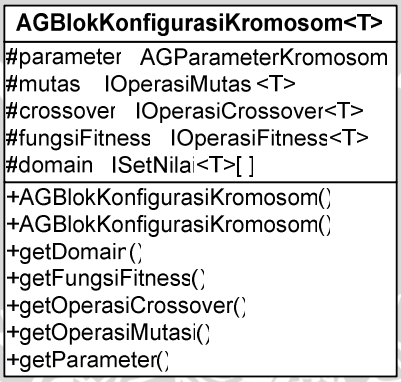
Setiap kromosom memiliki parameter yaitu probabilitas mutasi dan probabilitas *crossover*. Parameter kromosom direpresentasikan oleh kelas AGParameterKromosom.



Gambar 4.5 Diagram Kelas AGParameterKromosom

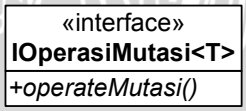
Kelas AGParameterKromosom mendefinisikan probabilitas mutasi dan probabilitas *crossover*. *Default* nilai parameter kromosom yaitu probabilitas mutasi 5% dan probabilitas crossover 80%.

Selain parameter, kromosom dapat memiliki data konfigurasi. Data konfigurasi kromosom meliputi parameter kromosom, metode mutasi, metode *crossover* dan fungsi fitness yang akan digunakan serta domain kromosom. Data-data konfigurasi tersebut biasanya sama untuk semua kromosom pada suatu populasi. Penyimpanan konfigurasi kromosom didefinisikan oleh kelas AGBlokKonfigurasiKromosom. Gambar 4.6 menunjukkan diagram kelas AGBlokKonfigurasiKromosom.



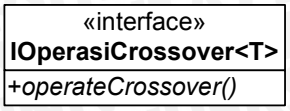
Gambar 4.6 Diagram Kelas AGBlokKonfigurasiKromosom

Operasi *crossover*, operasi mutasi dan fungsi *fitness* yang akan digunakan didefinisikan oleh kelas *interface* masing-masing operasi tersebut (IOperasiCrossover, IOperasiMutasi, dan IOperasiFitness). IOperasiMutasi merupakan *interface* untuk operasi mutasi. Implementasi operasi mutasi didefinisikan oleh kelas turunan dari IOperasiMutasi.



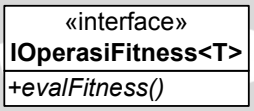
Gambar 4.7 Diagram Kelas *interface* IOperasiMutasi

IOperasiCrossover merupakan *interface* untuk operasi *crossover*. Implementasi operasi *crossover* didefinisikan oleh kelas turunan dari IOperasiCrossover.



Gambar 4.8 Diagram Kelas *interface* IOperasiCrossover

IOperasiFitness merupakan *interface* untuk fungsi *fitness*. Fungsi *fitness* / persamaan *fitness* didefinisikan oleh kelas turunan IOperasiFitness. Kelas turunan tersebut didefinisikan oleh *user library*.

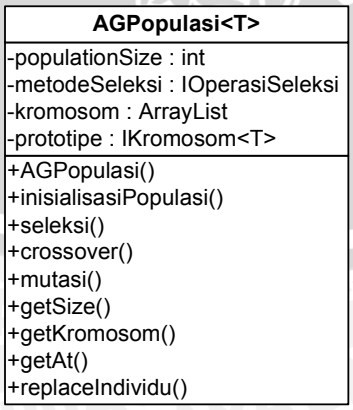


Gambar 4.9 Diagram Kelas *interface* IOperasiFitness

Interface IKromosom, kelas AGParameterKromosom, AGBlokKonfigurasiKromosom, *interface* IOperasiCrossover, IOperasiMutasi dan IOperasi Fitness tersebut dikelompokkan ke dalam *namespace* Kromosom.

4.2.2.1.2 Populasi

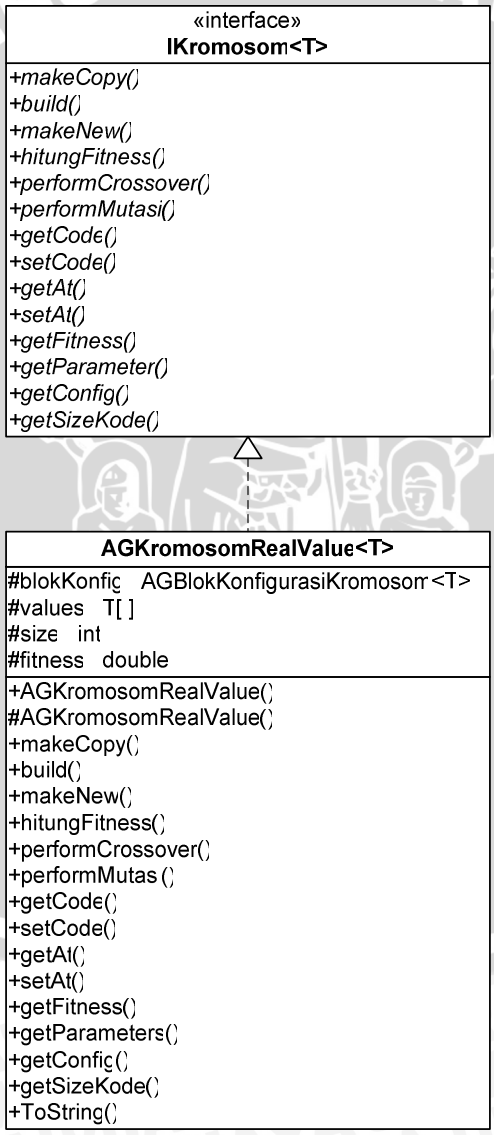
Objek populasi dalam *library* algoritma genetika ini direpresentasikan oleh kelas AGPopulasi. Objek populasi menyimpan objek kromosom dan konfigurasi populasi. Konfigurasi populasi meliputi parameter populasi yaitu ukuran populasi (*population size*) dan metode seleksi yang digunakan. AGPopulasi dikelompokkan ke dalam *namespace* Populasi. Gambar 4.10 menunjukkan diagram kelas AGPopulasi.



Gambar 4.10 Diagram Kelas AGPopulasi

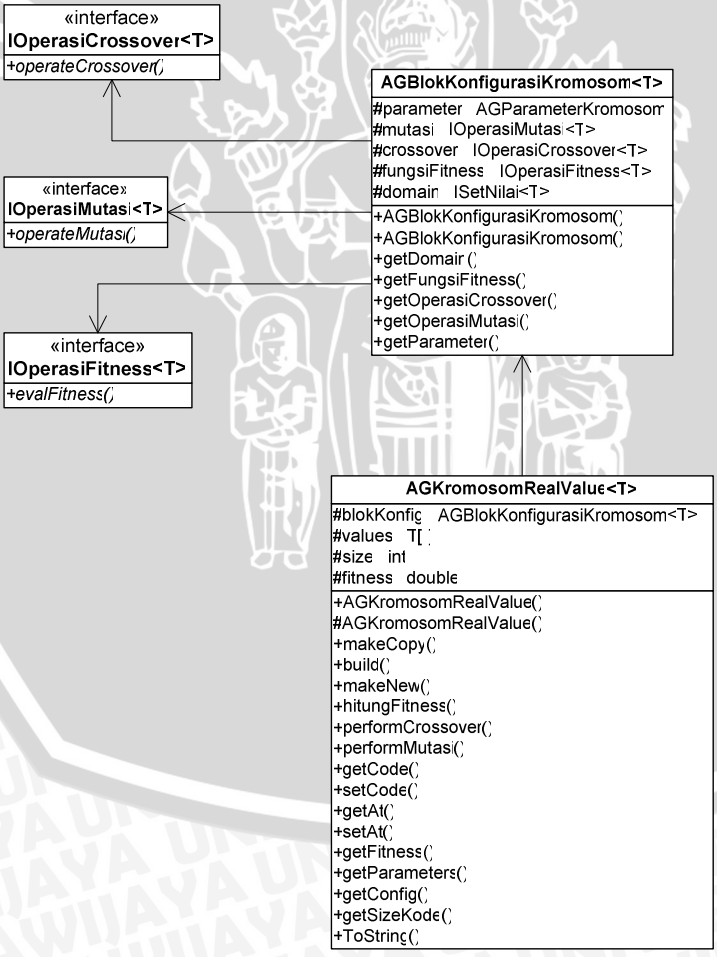
4.2.2.1.3 Representasi Kromosom

Interface kromosom, yang telah dibahas sebelumnya yaitu IKromosom, diimplementasikan oleh kelas turunannya. Implemetasi tersebut berupa representasi kromosom. Dalam algoritma genetika, kromosom dapat disajikan dalam beberapa bentuk sesuai dengan pengkodean yang digunakan yaitu pengkodean biner, bilangan riil atau pengkodean permutasi. Gambar 4.11 menunjukkan diagram kelas AGKromosomRealValue. Kelas AGKromosomRealValue dapat digunakan untuk kromosom yang mewakili solusi dengan pengkodean bilangan riil.



Gambar 4.11 Diagram Kelas AGKromosomRealValue

User library dapat mendefinisikan sendiri metode *crossover* ke dalam *method* `performCrossover()` atau metode mutasi ke dalam *method* `performMutasi()` pada kelas turunan yang mengimplementasikan *interface* IKromosom. Bila *user library* membuat kelas turunan dari AGKromosomRealValue maka metode mutasi dan *crossover* dapat didefinisikan dengan cara *override method* `performMutasi()` dan `performCrossover()`. *Library* algoritma genetika ini menyediakan alternatif lain mengenai hal tersebut. Operasi-operasi genetika tersebut, *crossover* dan mutasi, bisa didefinisikan dan diimplementasikan pada kelas-kelas secara terpisah. Selanjutnya, objek kelas-kelas tersebut disimpan pada AGBlokKonfigurasiKromosom. Gambar 4.12 menunjukkan relasi antara kelas AGKromosomRealValue dengan *interface* operasi – operasi genetika.

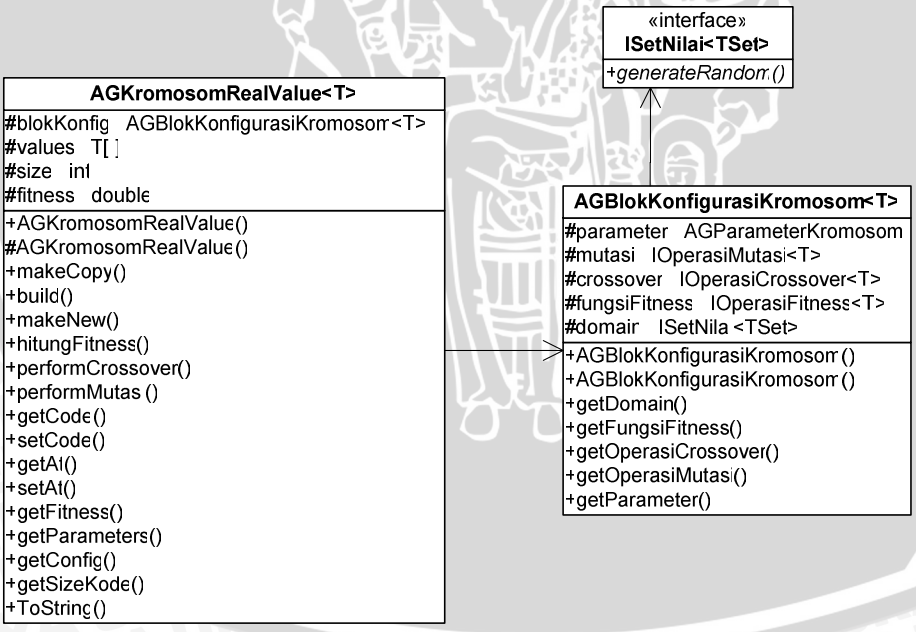


Gambar 4.12 Diagram Kelas Relasi antara Kelas AGKromosomRealValue dengan *interface* Operasi-operasi Genetika

Pada kebanyakan kasus yang menggunakan algoritma genetika, nilai gen-gen kromosom memiliki batasan-batasan (*constraint*) atau dapat disebut juga domain. Batasan-batasan ini pada *library* direalisasikan melalui objek set nilai. Kelas-kelas representasi kromosom yang merupakan implementasi *interface* IKromosom (misalnya: AGKromosomRealValue) menggunakan kelas AGBlokKonfigurasiKromosom untuk menyimpan objek set nilai yang mendefinisikan batasan-batasan nilai gen-gen. ISetNilai merupakan *interface* dari objek set nilai.

Gambar 4.13 Diagram Kelas *interface* ISetNilai

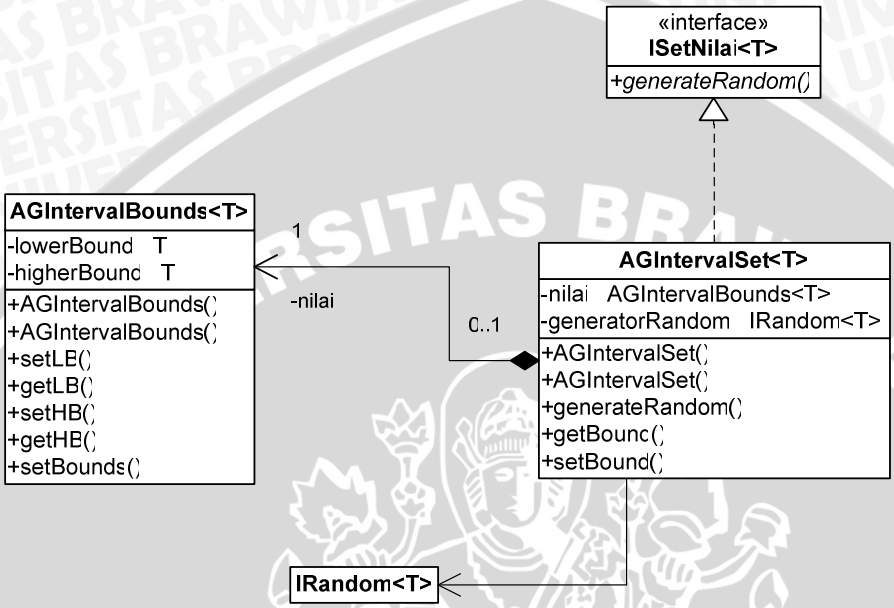
Gambar 4.14 menunjukkan relasi antara kelas AGKromosomRealValue, AGBlokKonfigurasiKromosom dan *interface* ISetNilai.



Gambar 4.14 Diagram Kelas Relasi antara AGKromosomRealValue, AGBlokKonfigurasiKromosom dan *interface* ISetNilai

Kelas AGIntervalSet, yang merupakan implentasi *interface* ISetNilai, merepresentasikan set nilai yang akan membangkitkan bilangan acak sesuai

dengan interval yang telah didefinisikan. Batas-batas dari interval didefinsikan oleh kelas AGIntervalBounds. *User library* harus menyediakan pembangkit (*generator*) bilangan acak yang mengimplementasikan *interface* IRandom.

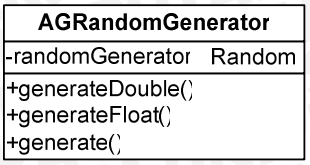


Gambar 4.15 Diagram Kelas AGIntervalSet dan AGIntervalBounds

Kelas AGKromosomRealValue, *interface* ISetNilai, kelas AGIntervalSet dan AG IntervalBounds tergabung di dalam *namespace* Representasi.

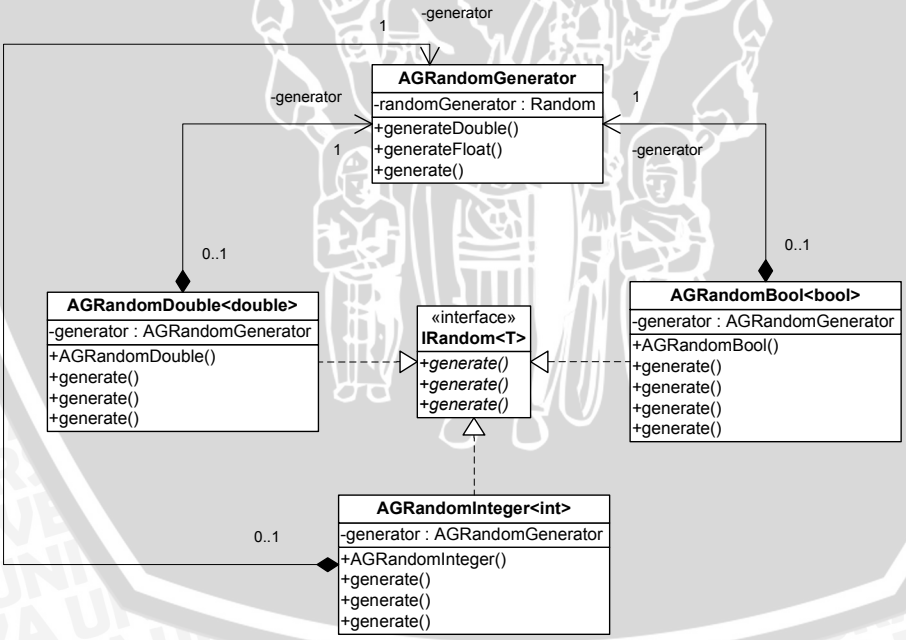
4.2.2.1.4 Pembangkit (*Generator*) Acak

Objek pembangkit acak bertugas membangkitkan bilangan acak. Tipe data bilangan acak dapat berupa *integer, float* dan *double*. Objek pembangkit acak direpresentasikan oleh kelas AGRandomGenerator. Bilangan acak yang dibangkitkan objek tersebut berinterval 0 sampai dengan 1 untuk tipe data *float* dan *double*, sedangkan untuk tipe data *integer* bilangan acak yang dibangkitkan antara 0 hingga nilai maksimum *integer* 32-bit (2.147.483.647). Gambar 4.16 menunjukkan diagram kelas AGRandomGenerator.



Gambar 4.16 Diagram Kelas AGRandomGenerator

Pada *library* algoritma genetika ini juga terdapat objek-objek pembangkit nilai acak yang lebih spesifik. Objek-objek ini yang akan menangani pembangkitan nilai acak dengan batasan nilai maksimum dan atau nilai minimum yang telah diberikan oleh *user library*. Objek-objek tersebut menggunakan AGRandomGenerator. Objek-objek pembangkit nilai acak yang lebih spesifik ini, yang juga merupakan implementasi *interface* IRandom, direpresentasikan oleh kelas AGRandomDouble (nilai acak dengan tipe data *double*), AGRandomInteger (nilai acak dengan tipe data *integer*), dan AGRandom Bool (nilai acak dengan tipe data *boolean*).

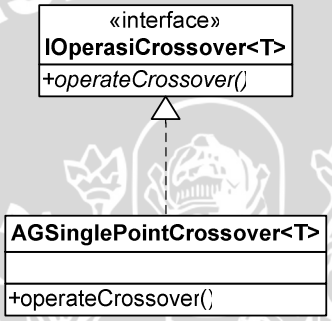


Gambar 4.17 Diagram Kelas Pembangkit Nilai Acak Beserta Relasinya Terhadap Kelas AGRandomGenerator

Objek-objek yang berkaitan dengan pembangkitan nilai acak tersebut dikelompokkan ke dalam *namespace* General.

4.2.2.1.5 Operasi *Crossover*

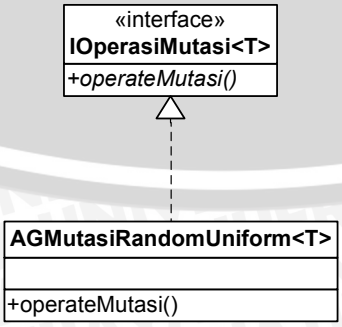
Metode *crossover* (pindah-silang) didefinisikan oleh objek operasi *crossover*. Objek tersebut merupakan implementasi dari *interface* IOperasiCrossover. Objek operasi *crossover* dikelompokkan ke dalam *namespace* OperasiCrossover. Gambar 4.18 menunjukkan diagram kelas AGSinglePointCrossover yang mendefinisikan metode pindah-silang *single point*.



Gambar 4.18 Diagram Kelas AGSinglePointCrossover

4.2.2.1.6 Operasi Mutasi

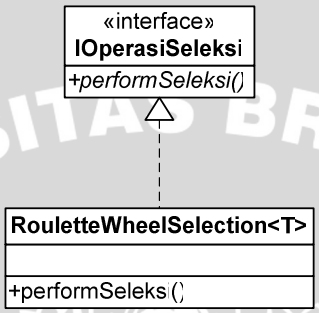
Metode mutasi didefinisikan oleh objek operasi mutasi. Objek operasi mutasi merupakan implementasi dari *interface* IOperasiMutasi. Objek tersebut dikelompokkan ke dalam *namespace* OperasiMutasi. Gambar 4.19 menunjukkan diagram kelas AGMutasiRandomUniform yang mendefinisikan metode mutasi *random uniform*.



Gambar 4.19 Diagram Kelas AGMutasiRandomUniform

4.2.2.1.7 Operasi Seleksi

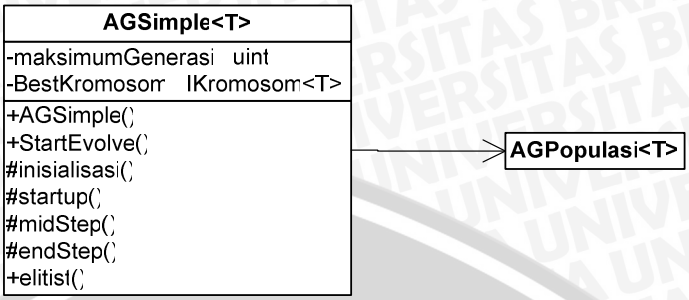
Seperti halnya metode *crossover* dan mutasi, metode seleksi didefinisikan oleh objek operasi seleksi. Objek operasi seleksi merupakan implementasi dari *interface* IOperasiSeleksi. Objek tersebut dikelompokkan ke dalam *namespace* OperasiSeleksi. Gambar 4.20 menunjukkan diagram kelas RouletteWheelSelection yang mendefinisikan metode seleksi *roulette-wheel*.



Gambar 4.20 Diagram Kelas RouletteWheelSelection

4.2.2.1.8 Algoritma

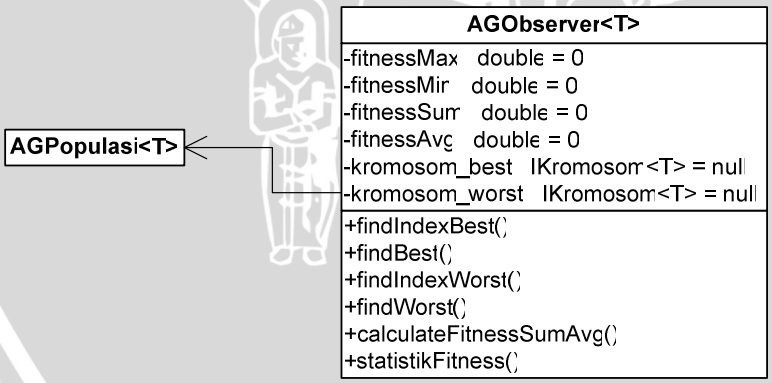
Objek algoritma genetika merupakan penghubung blok-blok objek yang telah dibahas sebelumnya. Objek tersebut mendefinisikan dan mengatur proses evolusi serta menentukan akhir dari proses evolusi. Objek algoritma genetika direpresentasikan oleh kelas AGSimple. Kelas AGSimple mengimplementasikan algoritma genetika simpek (*simple* GA) yaitu tanpa *overlap* populasi. *User library* perlu menyediakan objek populasi, seperti yang telah dibahas sebelumnya pada kelas AGPopulasi, sebagai populasi kromsom yang akan digunakan oleh kelas ini. Algoritma genetika ini mengimplementasikan elitisme. Kelas AGSimple dikelompokkan ke dalam *namespace* Algoritma.



Gambar 4.21 Diagram Kelas AGSimple Beserta Relasinya dengan Kelas AGPopulasi

4.2.2.1.9 Statistik Observer

Kelas AGObserver digunakan untuk melacak statistik populasi. Objek dari kelas ini menyimpan informasi-informasi statistik populasi yaitu kromosom terbaik (nilai *fitness* tertinggi), kromosom terburuk (nilai *fitness* terendah), *fitness* tertinggi (maksimum), *fitness* terendah (minimum), rata-rata *fitness*, dan total *fitness* dalam satu populasi. AGObserver dikelompokkan ke dalam *namespace* StatistikObserver.



Gambar 4.22 Diagram Kelas AGObserver Beserta Relasinya dengan Kelas AGPopulasi

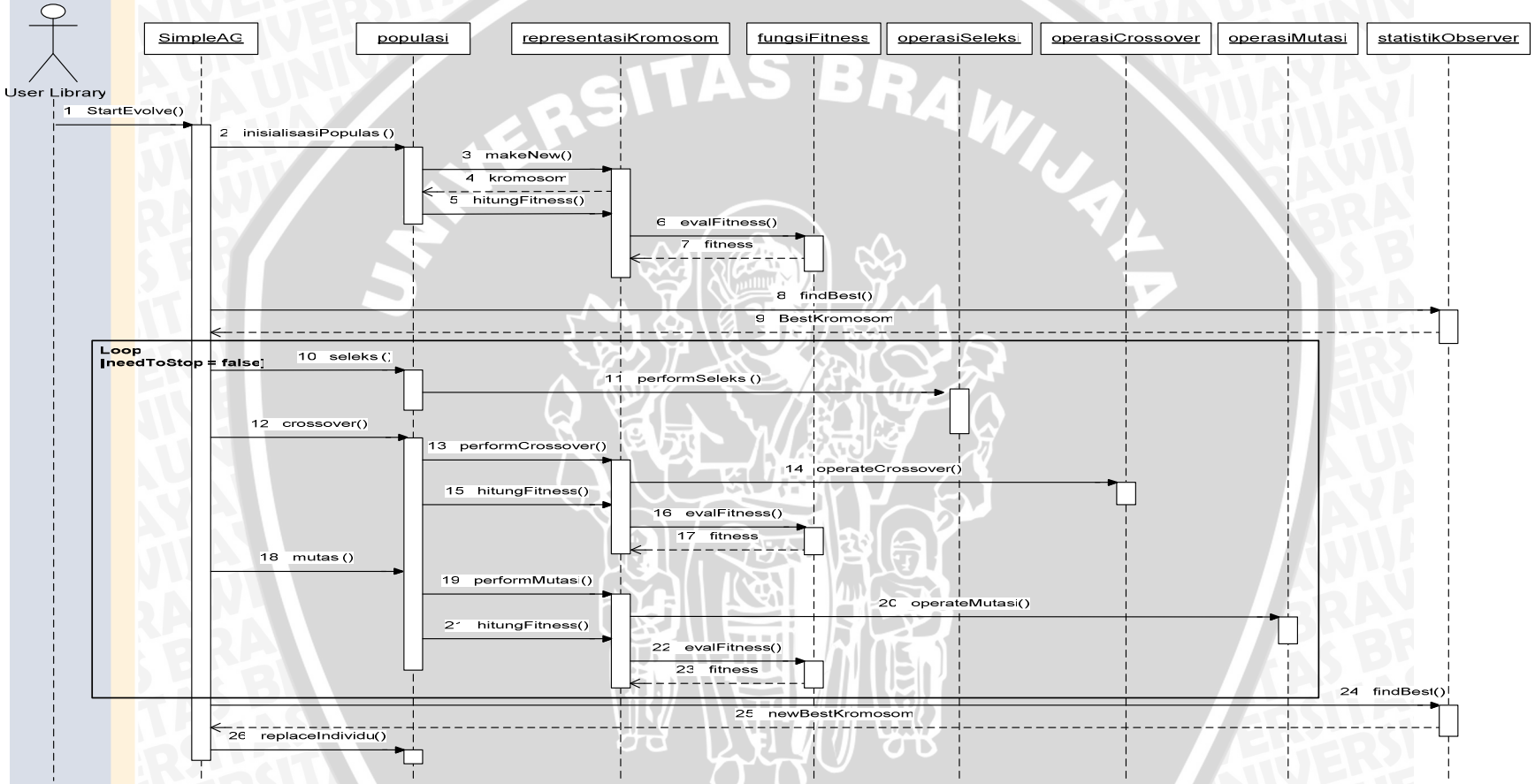
4.2.2.2 Diagram Sekuensial (*Sequence Diagram*)

Diagram sekuensial menggambarkan pemodelan urutan (*sequence*) proses interaksi antar objek yang disusun berdasarkan urutan waktu. Diagram

sekuensial dapat digunakan untuk menambah informasi kepada *use-case*. Diagram ini menunjukkan aktor yang terlibat pada interaksi, objek di dalam sistem dengan apa mereka berinteraksi dan operasi yang berhubungan dengan objek – objek ini.

- Diagram sekuensial Menjalankan Algoritma

Gambar 4.23 merupakan diagram sekuensial Menjalankan Algoritma. Diagram sekuensial ini menggambarkan interaksi *user library* ketika menjalankan proses evolusi algortima genetika untuk mencari solusi permasalahan yang telah didefinisikan sebelumnya.



Gambar 4.23 Diagram Sekuensial Menjalankan Algoritma

4.3 Implementasi

Pada bab ini dibahas mengenai implementasi perangkat lunak berdasarkan perancangan perangkat lunak yang telah dibuat. Pembahasan terdiri dari lingkungan implementasi (meliputi spesifikasi perangkat keras serta perangkat lunak), dan pemaparan (dokumentasi) detil kelas-kelas yang terdapat dalam *library* algoritma genetika, yaitu fungsi kelas dalam *library* serta prosedur (*method*) dan atribut yang dimilikinya.

4.3.1 Lingkungan Implementasi

Library algoritma genetika ini (AG3NLib) dibuat menggunakan bahasa pemrograman C#. *Library* diimplementasikan dengan spesifikasi yang ditunjukkan pada tabel 4.11 dan 4.12.

Tabel 4.11 Spesifikasi Perangkat Keras Komputer

Nama Komponen	**Spesifikasi**
Prosesor	Intel Pentium Dual-Core E2180 2,00 GHz
Memori (RAM)	1024 MB

Tabel 4.12 Spesifikasi Perangkat Lunak

Sistem Operasi	Windows XP Service Pack 3
Bahasa Pemrograman	C#
Lingkungan Pemrograman	.NET Framework 4.0
IDE	Microsoft Visual Studio 2010

4.3.2 Implementasi Kelas

Kelas-kelas yang telah dirancang pada proses perancangan direalisasikan pada suatu *file* program *.cs. Tabel 4.13 menunjukkan pasangan antara kelas dan file program yang berisi implementasinya yang dikelompokkan sesuai dengan *namespace* masing-masing kelas.

Tabel 4.13 Implementasi Kelas pada Kode Program *.cs

Namespace	**Nama Kelas / *Interface***	**Nama File Program**
Algoritma	AGSimple	SimpleAG.cs
General	AGRandomBool	AGRandom.cs
	AGRandomDouble	
	AGRandomGenerator	
	AGRandomInteger	
	IRandom	
Kromosom	AGBlokKonfigurasiKromosom	AGKromosom.cs
	AGParameterKromosom	
	IKromosom	
	IOperasiCrossover	AGOperasiKromosom.cs
	IOperasiFitness	
	IOperasiMutasi	
Kromosom.OperasiCrossover	AGSinglePointCrossover	AGCrossover.cs
Kromosom.OperasiMutasi	AGMutasiRandomUniform	AGMutasi.cs
Kromosom.Representasi	AGKromosomRealValue	AGKromosomRealValue.cs
	AGIntervalBounds	AGSetNilai.cs
	AGIntervalSet	
	ISetNilai	
Populasi	AGPopulasi	AGPopulasi.cs
	IOperasiSeleksi	AGOperasiPopulasi.cs
Populasi.OperasiSeleksi	RouletteWheelSelection	AGSeleksi.cs
StatistikObserver	AGObserver	AGObserver.cs

4.3.2.1 *Interface* IKromosom

a). **Deskripsi**

Interface IKromosom merupakan *interface* untuk representasi kromosom dalam *library* ini. IKromosom hanya berisi *signature* fungsi anggota yang digunakan oleh kelas implementasinya. Semua kelas turunan *interface* IKromosom harus mengimplementasikan semua fungsi anggota yang dimiliki IKromosom. Parameter *T* pada kelas ini merupakan tipe data yang digunakan oleh domain kromosom.

b). **Fungsi Anggota**

```
IKromosom<T> makeCopy()
```

Method ini menginstansiasi objek kromosom salinan atau bisa dikatakan kromosom kloning. Kromosom yang dibuat merupakan kloning, jadi atribut yang dimilikinya sama persis dengan objek kromosom yang dikloning.

Kembalian:

Kembalian dari *method* ini yaitu objek kromosom yang memiliki atribut sama persis dengan objek kromosom yang dikloning.

```
void build()
```

Method ini membangkitkan (*generate*) nilai gen-gen yang menyusunnya.

```
IKromosom<T> makeNew()
```

Method ini menginstansiasi objek kromosom baru. Konfigurasi kromosom baru mengikuti prototipe objek kromosom yang telah diinstansiasi oleh *user library*.

Kembalian:

Kembalian dari *method* ini yaitu objek kromosom baru yang memiliki konfigurasi dan ukuran kromosom yang sama dengan objek kromosom prototipe.

```
void hitungFitness()
```

Method ini menghitung *fitness* kromosom kemudian disimpan dalam variabel.

```
void performCrossover(ref IKromosom<T> kromosomPertama, ref
IKromosom<T> kromosomKedua)
```

Method ini dipanggil ketika pindah-silang (*crossover*) harus dilakukan antara 2 kromosom induk.

Parameter:

`kromosomPertama` *reference variable* objek kromosom *parent* 1

`kromosomKedua` *reference variable* objek kromosom *parent* 2

```
void performMutasi(ref IKromosom<T> offspring)
```

Method ini dipanggil ketika mutasi harus dilakukan oleh kromosom.

Parameter:

`offspring` *reference variable* objek kromosom yang akan mengalami mutasi.

```
T[] getCode()
```

Method ini memberikan kembalian kode kromosom (nilai gen-gen).

Kembalian:

Kembalian dari *method* ini yaitu kode kromosom (nilai gen-gen).

```
void setCode(T[] values)
```

Method ini memberikan nilai gen-gen kromosom sesuai dengan nilai parameter yang diberikan.

Parameter:

`values` nilai gen-gen pada suatu kromosom berupa *array*

```
T getAt(int pos)
```

Method ini mengembalikan nilai gen kromosom pada posisi gen yang telah ditentukan.

Parameter:

`pos` posisi gen

Kembalian:

Kembalian dari *method* ini yaitu nilai gen kromosom pada posisi gen yang telah ditentukan.

```
void setAt(T value, int pos)
```

Method ini memberikan (*set*) nilai gen kromosom pada posisi yang diberikan.

Parameter:

value nilai gen

pos posisi gen

```
double getFitness()
```

Method ini mengembalikan nilai *fitness* objek kromosom.

Kembalian:

Kembalian dari *method* ini yaitu nilai *fitness* objek kromosom.

```
AGParameterKromosom getParameters()
```

Method ini mengembalikan parameter kromosom.

Kembalian:

Kembalian dari *method* ini yaitu parameter kromosom.

```
AGBlokKonfigurasiKromosom<T> getConfig()
```

Method ini mengembalikan data konfigurasi kromosom.

Kembalian:

Kembalian dari *method* ini yaitu data konfigurasi kromosom.

```
int getSizeKode()
```

Method ini mengembalikan ukuran kromosom.

Kembalian:

Kembalian dari *method* ini yaitu ukuran/panjang kromosom.

Tabel 4.14 Fungsi Anggota *Interface* IKromosom.

4.3.2.2 Kelas AGParameterKromosom

a). **Deskripsi**

Kelas ini berfungsi menyimpan parameter-parameter kromosom. Parameter kromosom yang disimpan yaitu probabilitas mutasi dan probabilitas *crossover*.

b). **Atribut**

Nama	Deskripsi
`protected double probMutasi`	Probabilitas mutasi memiliki interval nilai antara 0 hingga 1.
`protected double probCrossover`	Probabilitas *crossover* memiliki interval nilai antara 0 hingga 1.

Tabel 4.15 Atribut Kelas AGParameterKromosom

c). **Fungsi Anggota**

```
public AGParameterKromosom(double probabilitasMutasi, double
probabilitasCrossover)
```

Konstruktor ini menginisialisasi parameter sesuai dengan nilai yang didefinisikan *user library*.

Parameter:

probabilitasMutasi probabilitas mutasi dengan interval antara 0 hingga 1

probabilitasCrossover probabilitas *crossover* dengan interval 0 hingga 1

```
public AGParameterKromosom()
```

Method ini menginisialisasi parameter dengan nilai *default* yaitu probabilitas mutasi 5% dan probabilitas crossover 80%.

```
public double getProbabilitasMutasi()
```

Method ini mengembalikan probabilitas mutasi.

Kembalian:

Kembalian dari *method* ini yaitu probabilitas mutasi.

```
public void setProbabilitasMutasi(double prob)
```

Method ini memberikan (*set*) probabilitas mutasi.

Parameter:

`prob` probabilitas mutasi. Nilainya berinterval antara 0 hingga 1.

```
public double getProbabilitasCrossover()
```

Method ini mengembalikan probabilitas *crossover*.

Kembalian:

Kembalian dari *method* ini yaitu probabilitas *crossover*.

```
public void setProbabilitasCrossover(double prob)
```

Method ini memberikan (*set*) probabilitas *crossover*.

Parameter:

`prob` probabilitas *crossover*. Nilainya berinterval antara 0 hingga 1.

Tabel 4.16 Fungsi Anggota Kelas AGParameterKromosom

4.3.2.3 Kelas AGBlokKonfigurasiKromosom

a). **Deskripsi**

Kelas ini menyimpan data konfigurasi kromosom. Data konfigurasi kromosom meliputi yaitu parameter kromosom, metode mutasi, metode *crossover* dan fungsi fitness yang akan digunakan serta domain kromosom. Parameter *T* pada kelas ini merupakan tipe data yang digunakan oleh domain kromosom.

b). **Atribut**

Nama	Deskripsi
`protected AGParameterKromosom parameter`	Objek parameter kromosom yang telah diinstansiasi sebelumnya oleh *user library*.
`protected IOperasiMutasi<T> mutasi`	Objek operasi mutasi yang telah diinstansiasi sebelumnya oleh *user library*.
`protected IOperasiCrossover<T> crossover`	Objek operasi *crossover* yang telah diinstansiasi sebelumnya oleh *user library*.
`protected IOperasiFitness<T> fungsiFitness`	Objek fungsi *fitness* yang telah diinstansiasi sebelumnya oleh *user library*.
`protected ISetNilai<T>[] domain`	Objek domain yang telah diinstansiasi sebelumnya oleh *user library*.

Tabel 4.17 Atribut Kelas AGBlokKonfigurasiKromosom

c). **Fungsi Anggota**

```
public AGBlokKonfigurasiKromosom(ISetNilai<T>[] Domain,
            int DomainCount,
            AGParameterKromosom Parameter,
            IOperasiCrossover<T> Crossover,
            IOperasiMutasi<T> Mutasi,
            IOperasiFitness<T> FungsiFitness)
```

Konstruktor ini menginisialisasi data konfigurasi kromosom. Konstruktor ini digunakan bila domain yang diberikan berupa *array* (lebih dari satu domain).

Parameter:

`Domain` domain merupakan batasan kromosom berupa set nilai

`DomainCount` jumlah domain yang diberikan

`Parameter` objek parameter kromosom yang telah diinstantiasi sebelumnya

`Crossover` objek operasi *crossover* yang telah diinstansiasi sebelumnya

`Mutasi` objek operasi mutasi yang telah diinstansiasi sebelumnya

`FungsiFitness` objek fungsi *fitness* yang telah diinstansiasi sebelumnya

```
public AGBlokKonfigurasiKromosom(ISetNilai<T> Domain,
            AGParameterKromosom Parameter,
            IOperasiCrossover<T> Crossover,
            IOperasiMutasi<T> Mutasi,
            IOperasiFitness<T> FungsiFitness)
```

Konstruktor ini menginisialisasi data konfigurasi kromosom. Konstruktor ini digunakan bila domain yang diberikan hanya satu domain.

Parameter:

`Domain` domain merupakan batasan kromosom berupa set nilai

`Parameter` objek parameter kromosom yang telah diinstantiasi sebelumnya

`Crossover` objek operasi *crossover* yang telah diinstansiasi sebelumnya

`Mutasi` objek operasi mutasi yang telah diinstansiasi sebelumnya

`FungsiFitness` objek fungsi *fitness* yang telah diinstansiasi sebelumnya

```
public ISetNilai<T> getDomain(int index)
```

Method ini mengembalikan domain kromosom untuk posisi/indeks yang telah ditentukan.

Parameter:

`index` posisi/indeks array domain kromosom yang telah disimpan

Kembalian:

Kembalian dari *method* ini yaitu domain kromsom.

```
public IOperasiFitness<T> getFungsiFitness()
```

Method ini mengembalikan objek fungsi *fitness*.

Kembalian:

Kembalian dari *method* ini yaitu objek fungsi *fitness*.

```
public IOperasiCrossover<T> getOperasiCrossover()
```

Method ini mengembalikan objek operasi *crossover*.

Kembalian:

Kembalian dari *method* ini yaitu objek operasi *crossover*.

```
public IOperasiMutasi<T> getOperasiMutasi()
```

Method ini mengembalikan objek operasi mutasi.

Kembalian:

Kembalian dari *method* ini yaitu objek operasi mutasi.

```
public AGParameterKromosom getParameter()
```

Method ini mengembalikan objek parameter kromosom.

Kembalian:

Kembalian dari *method* ini yaitu objek parameter kromosom.

Tabel 4.18 Fungsi Anggota Kelas AGBlokKonfigurasiKromosom

4.3.2.4 *Interface* IOperasiMutasi

a). **Deskripsi**

Interface IOperasiMutasi merupakan *interface* untuk operasi mutasi. IOperasiMutasi hanya berisi *signature* fungsi anggota yang digunakan oleh kelas implementasinya. Semua kelas turunan *interface* IOperasiMutasi harus mengimplementasikan semua fungsi anggota yang dimiliki IOperasiMutasi. Parameter *T* pada kelas ini merupakan tipe data yang digunakan oleh domain kromosom.

b). **Fungsi Anggota**

```
void operateMutasi(ref IKromosom<T> offspring)
```

Method ini menjalankan operasi mutasi.

Parameter:

`offspring` *reference variable* objek kromosom yang akan mengalami mutasi.

Tabel 4.19 Fungsi Anggota *Interface* IOperasiMutasi

4.3.2.5 *Interface* IOperasiCrossover

a). **Deskripsi**

Interface IOperasiCrossover merupakan *interface* untuk operasi *crossover*. IOperasiCrossover hanya berisi *signature* fungsi anggota yang digunakan oleh kelas implementasinya. Semua kelas turunan *interface* IOperasiCrossover harus mengimplementasikan semua fungsi anggota yang dimiliki IOperasiCrossover. Parameter *T* pada kelas ini merupakan tipe data yang digunakan oleh domain kromosom.

b). **Fungsi Anggota**

```
void operateCrossover(ref IKromosom<T> parent1, ref IKromosom<T>
parent2)
```

Method ini menjalankan operasi *crossover*.

Parameter:

`parent1` kromosom induk 1

`parent2` kromosom induk 2

Tabel 4.20 Fungsi Anggota *Interface* IOperasiCrossover

4.3.2.6 *Interface* IOperasiFitness

a). **Deskripsi**

Interface IOperasiFitness merupakan *interface* untuk fungsi *fitness*. IOperasiFitness hanya berisi *signature* fungsi anggota yang digunakan oleh kelas implementasinya. Semua kelas turunan *interface* IOperasiFitness harus mengimplementasikan semua fungsi anggota yang dimiliki IOperasiFitness. Kelas turunan tersebut berisi persamaan/fungsi *fitness* yang didefinisikan oleh *user library*. Parameter *T* pada kelas ini merupakan tipe data yang digunakan oleh domain kromosom.

b). **Fungsi Anggota**

```
double evalFitness(IKromosom<T> kromosom)
```

Method ini menghitung nilai fitness kromosom.

Parameter:

`kromosom` objek kromosom yang akan dihitung *fitness*nya

Kembalian:

Kembalian dari *method* ini yaitu nilai *fitness* kromosom.

Tabel 4.21 Fungsi Anggota *Interface* IOperasiFitness

4.3.2.7 Kelas AGKromosomRealValue

a). **Deskripsi**

Kelas AGKromosomRealValue ini merupakan representasi kromosom dengan pengkodean bilangan riil. Kelas ini adalah turunan dan mengimplementasikan *interface* IKromosom. Parameter *T* pada kelas ini merupakan tipe data yang digunakan oleh domain kromosom.

b). **Atribut**

Nama	Deskripsi
`protected AGBlokKonfigurasiKromosom<T> blokKonfig`	Objek blok konfigurasi kromosom.
`protected T[] values`	Nilai kode kromosom (gen).
`protected int size`	Ukuran/panjang kromosom atau bisa disebut jumlah gen dalam kromosom.
`protected double fitness`	Nilai *fitness* kromosom.

Tabel 4.22 Atribut Kelas AGKromosomRealValue

c). **Fungsi Anggota**

```
public AGKromosomRealValue(AGBlokKonfigurasiKromosom<T>
BlokKonfig, int Size)
```

Konstruktor ini menginisialisasi konfigurasi kromosom serta memanggil *method* build() untuk *generate* nilai gen-gen yang menyusunnya.

```
protected AGKromosomRealValue(AGKromosomRealValue<T> source)
```

Konstruktor ini berfungsi menyalin data-data kromosom (konfigurasi kromosom, ukuran kromosom, gen, *fitness*) yang akan dikloning. Konstruktor ini hanya digunakan oleh *method* makeCopy() untuk menginstansiasi kromosom kloning.

```
public virtual IKromosom<T> makeCopy()
```

Method ini menginstansiasi objek kromosom salinan atau bisa dikatakan kromosom kloning. Kromosom yang dibuat merupakan kloning, atribut yang dimilikinya sama persis dengan objek kromosom yang dikloning.

Kembalian:

Kembalian dari *method* ini yaitu objek kromosom yang memiliki atribut sama persis dengan objek kromosom yang dikloning.

```
public virtual void build()
```

Method ini membangkitkan (*generate*) nilai gen-gen yang menyusunnya.

```
public virtual IKromosom<T> makeNew()
```

Method ini menginstansiasi objek kromosom baru. Konfigurasi kromosom baru mengikuti prototipe objek kromosom yang telah diinstansiasi oleh *user library*.

Kembalian:

Kembalian dari *method* ini yaitu objek kromosom baru yang memiliki konfigurasi dan ukuran kromosom yang sama dengan objek kromosom prototipe.

```
public void hitungFitness()
```

Method ini menghitung *fitness* kromosom kemudian disimpan dalam variable *fitness*.

```
public virtual void performCrossover(ref IKromosom<T>
kromosomPertama, ref IKromosom<T> pair)
```

Method ini dipanggil ketika pindah-silang (*crossover*) harus dilakukan antara 2 kromosom induk.

Parameter:

`kromosomPertama` *reference variable* objek kromosom *parent* pertama

`pair` *reference variable* objek kromosom pasangan (*parent* kedua)

```
public virtual void performMutasi(ref IKromosom<T> _offspring)
```

Method ini dipanggil ketika mutasi harus dilakukan oleh kromosom.

Parameter:

`_offspring` *reference variable* objek kromosom yang akan mengalami mutasi.

```
public T[] getCode()
```

Method ini memberikan kembalian kode kromosom (nilai gen yang menyusun kromosom).

Kembalian:

Kembalian dari *method* ini yaitu kode kromosom (nilai gen-gen).

```
public void setCode(T[] Values)
```

Method ini memberikan (*set*) nilai gen-gen kromosom sesuai dengan nilai parameter yang diberikan.

Parameter:

`Values` nilai gen-gen pada suatu kromosom berupa *array*

```
public T getAt(int pos)
```

Method ini mengembalikan nilai gen kromosom pada posisi gen yang telah ditentukan.

Parameter:

pos posisi gen

Kembalian:

Kembalian dari *method* ini yaitu nilai gen kromosom pada posisi gen yang telah ditentukan.

```
public void setAt(T value, int pos)
```

Method ini memberikan (*set*) nilai gen kromosom pada posisi yang diberikan.

Parameter:

value nilai gen

pos posisi gen

```
public double getFitness()
```

Method ini mengembalikan nilai *fitness* objek kromosom.

Kembalian:

Kembalian dari *method* ini yaitu nilai *fitness* objek kromosom.

```
public AGParameterKromosom getParameters()
```

Method ini mengembalikan parameter kromosom.

Kembalian:

Kembalian dari *method* ini yaitu parameter kromosom.

```
public AGBlokKonfigurasiKromosom<T> getConfig()
```

Method ini mengembalikan data konfigurasi kromosom.

Kembalian:

Kembalian dari *method* ini yaitu data konfigurasi kromosom.

```
public int getSizeKode()
```

Method ini mengembalikan ukuran kromosom.

Kembalian:

Kembalian dari *method* ini yaitu ukuran/panjang kromosom.

```
public override string ToString()
```

Method ini merupakan *overloading* terhadap *method* ToString() yang terdapat dalam *library mscorlib.dll*. *Method* ini mengkonversi nilai gen untuk ditampilkan sebagai *string*.

Kembalian:

Kembalian dari *method* ini yaitu objek nilai gen dalam bentuk *string*.

Tabel 4.23 Fungsi Anggota Kelas AGKromosomRealValue

4.3.2.8 *Interface* ISetNilai

a). **Deskripsi**

Interface ISetNilai merupakan *interface* untuk membangkitkan (*generate*) nilai acak yang dipilih dari set batasan (*constraint*) kromosom yang telah didefinisikan. Implementasi ISetNilai didefinisikan oleh kelas turunan *interface* ISetNilai. Parameter *T* pada kelas ini merupakan tipe data batasan (*bounds*).

b). **Fungsi Anggota**

```
T generateRandom()
```

Method ini mengembalikan nilai acak yang dipilih dari set.

Tabel 4.24 Fungsi Anggota *Interface* ISetNilai

4.3.2.9 Kelas AGIntervalSet

a). **Deskripsi**

Kelas AGIntervalSet ini merepresentasikan set nilai yang mempunyai interval. Interval ditentukan oleh batas-batasnya. Kelas ini menggunakan pembangkit (*generator*) nilai acak yang telah ditentukan *user library* untuk membangkitkan nilai acak dengan interval tersebut. Kelas ini merupakan turunan

dan mengimplementasikan *interface* ISetNilai. Parameter *T* pada kelas ini merupakan tipe data batasan (*bounds*).

b). **Atribut**

Nama	Deskripsi
`private AGIntervalBounds<T> nilai`	Objek batas-batas interval.
`IRandom<T> generatorRandom`	Pembangkit nilai acak.

Tabel 4.25 Atribut Kelas AGIntervalSet

c). **Fungsi Anggota**

```
public AGIntervalSet(AGIntervalBounds<T> Nilai, IRandom<T>
GeneratorRandom)
```

Konstruktor ini menginisialisasi set nilai dengan batas-batas dan pembangkit nilai acak.

Parameter:

`Nilai` batas-batas interval

`GeneratorRandom` pembangkit nilai acak yang digunakan

```
public AGIntervalSet(IRandom<T> GeneratorRandom)
```

Konstruktor ini menginisialisasi pembangkit nilai acak namun tanpa mendefinisikan batasan interval.

Parameter:

`GeneratorRandom` pembangkit nilai acak yang digunakan

```
public virtual T generateRandom()
```

Method ini mengembalikan nilai acak yang dipilih dari set.

Parameter:

`GeneratorRandom` pembangkit nilai acak yang digunakan

```
public AGIntervalBounds<T> getBound()
```

Method ini mengembalikan batasan interval.

Kembalian:

Kembalian dari *method* ini yaitu batasan interval.

```
public void setBound(AGIntervalBounds<T> Nilai)
```

Method ini mengeset batasan interval.

Parameter:

Nilai batas-batas interval

Tabel 4.26 Fungsi Anggota Kelas AGIntervalSet

4.3.2.10 Kelas AGIntervalBounds

a). **Deskripsi**

Kelas AGIntervalBounds merepresentasikan set nilai batas-batas interval. Parameter *T* pada kelas ini merupakan tipe data batasan (*bounds*).

b). **Atribut**

Nama	Deskripsi
`private T lowerBound`	Batas bawah.
`private T higherBound`	Batas atas.

Tabel 4.27 Atribut Kelas AGIntervalBounds

c). **Fungsi Anggota**

```
public AGIntervalBounds(T lower, T higher)
```

Konstruktor ini menginisialisasi batas-batas yaitu batas atas dan batas bawah yang nilainya diberikan oleh *user library*.

Parameter:

lower nilai batas bawah

higher nilai batas atas

```
public void setLB(T lower)
```

Method ini mengeset nilai batas bawah.

Parameter:

lower nilai batas bawah

```
public T getLB()
```

Method ini mengembalikan nilai batas bawah.

Kembalian:

Kembalian dari *method* ini yaitu nilai batas bawah.

```
public void setHB(T higher)
```

Method ini mengeset nilai batas atas.

Parameter:

higher nilai batas atas

```
public T getHB()
```

Method ini mengembalikan nilai batas atas.

Kembalian:

Kembalian dari *method* ini yaitu nilai batas atas.

```
public void setBounds(T lower, T higher)
```

Method ini mengeset nilai batas bawah dan batas atas.

Parameter:

lower nilai batas bawah

higher nilai batas atas

Tabel 4.28 Fungsi Anggota Kelas AGIntervalBounds

4.3.2.11 Kelas AGPopulasi

a). **Deskripsi**

Kelas AGPopulasi digunakan untuk menyimpan sekumpulan objek kromosom. Operasi-operasi genetika yaitu seleksi, *crossover* dan mutasi diatur oleh kelas ini. Parameter *T* pada kelas ini merupakan tipe data yang digunakan oleh domain kromosom.

b). **Atribut**

Nama	Deskripsi
`private int populationSize`	Ukuran populasi.
`private IOperasiSeleksi metodeSeleksi`	Objek operasi seleksi yaitu metode seleksi yang digunakan yang telah diinstansiasi sebelumnya oleh *user library*.
`private ArrayList kromosom`	Sekumpulan objek kromosom yang disimpan dalam *arraylist*. Objek prototipe kromosom selanjutnya dipakai untuk menciptakan kromosom-kromosom baru. Kromosom baru tersebut disimpan pada variabel ini.
`private IKromosom<T> prototype`	Objek prototipe kromosom yang telah diinstansiasi sebelumnya oleh *user library*. Instansiasi kromosom baru menggunakan prototipe ini.

Tabel 4.29 Atribut Kelas AGPopulasi

c). **Fungsi Anggota**

`public AGPopulasi(IKromosom<T> Prototipe, IOperasiSeleksi MetodeSeleksi, int PopulationSize)`
Konstruktor ini menginisialisasi objek prototipe kromosom, objek operasi

seleksi

Parameter:

`Prototipe` objek prototipe kromosom yang telah diinstansiasi sebelumnya oleh *user library*

`MetodeSeleksi` objek operasi seleksi yaitu metode seleksi yang digunakan yang telah diinstansiasi sebelumnya oleh *user library*

`PopulationSize` ukuran populasi

```
public void inisialisasiPopulasi()
```

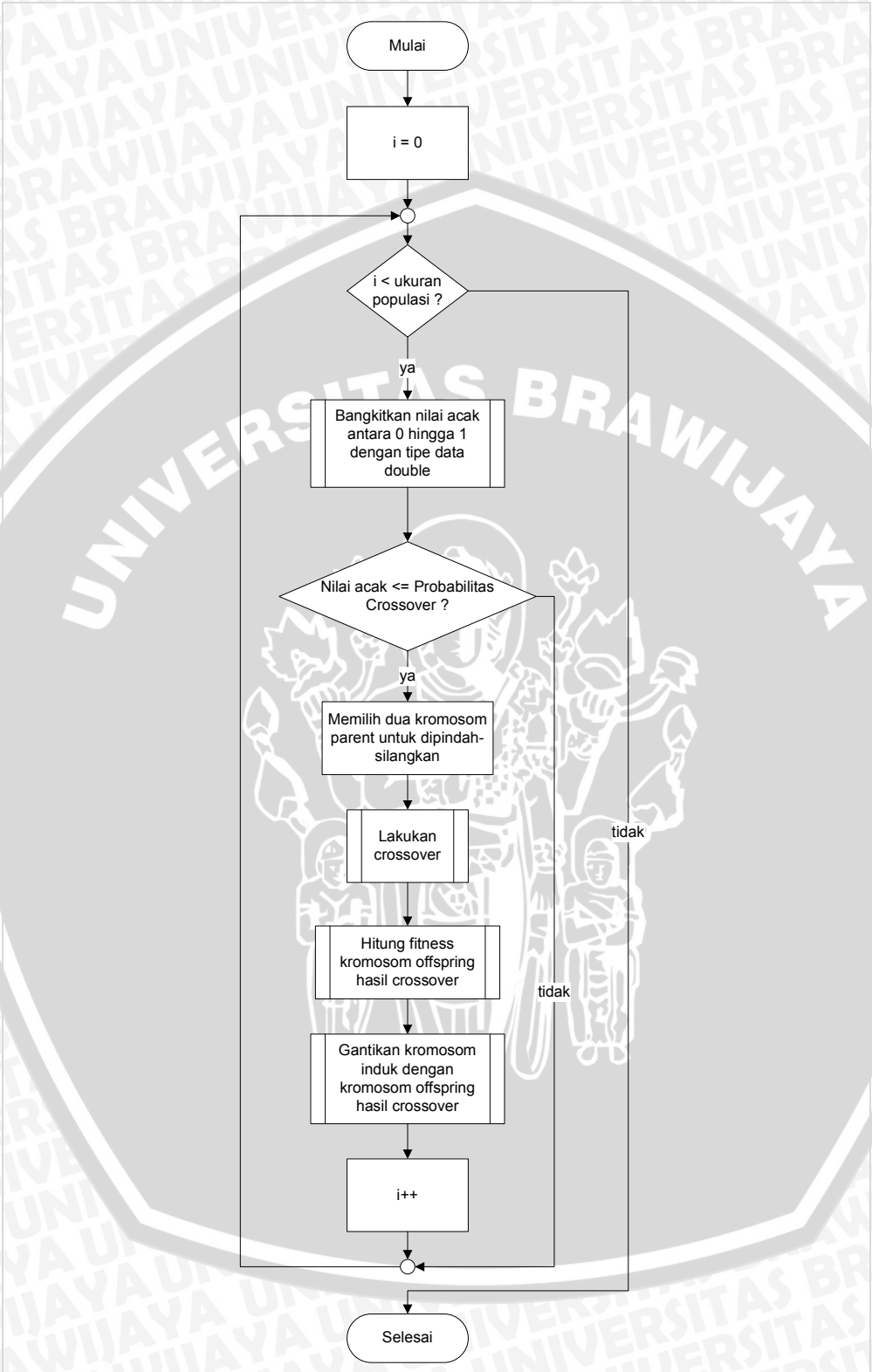
Method ini digunakan untuk *generate* populasi baru. Populasi baru berisi kumpulan-kumpulan kromosom sebanyak ukuran populasi. Objek kromosom baru yang diinstansiasi kemudian dihitung nilai *fitness*nya dan disimpan ke dalam *arraylist*.

```
public virtual void seleksi()
```

Method ini memanggil *method* `performSeleksi()` pada objek operasi seleksi untuk menjalankan operasi seleksi. Objek operasi seleksi disimpan pada variabel metodeSeleksi.

```
public virtual void crossover()
```

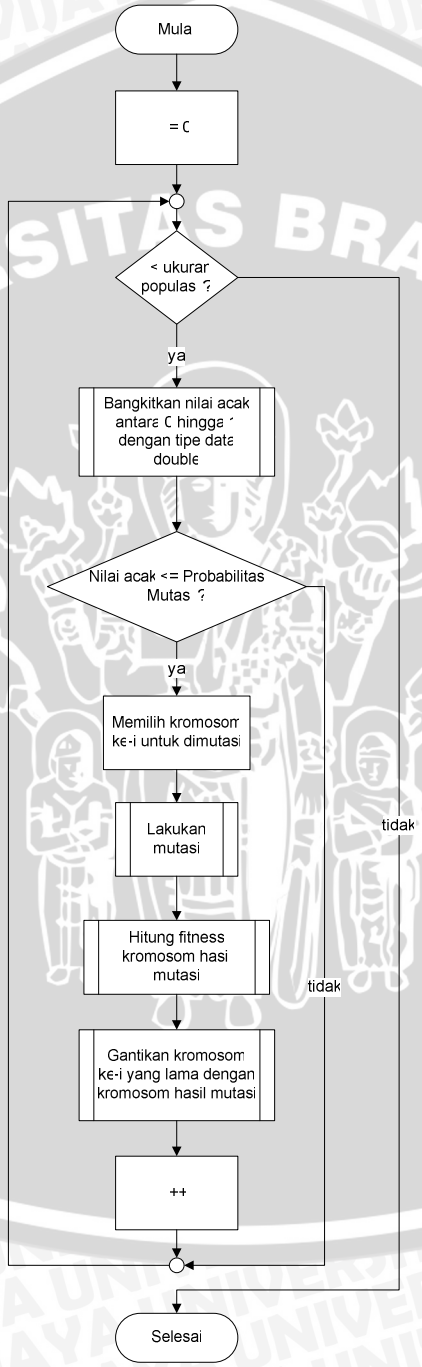
Method ini mengatur operasi *crossover*. Probabilitas *crossover* menentukan peluang *crossover* terjadi atau tidak. *Method* `performCrossover()` yang terdapat pada objek prototipe kromosom yang akan menjalankan proses *crossover*.



Gambar 4.24 Diagram Alir Prosedur `crossover` pada Kelas AGPopulasi

```
public virtual void mutasi()
```

Method ini mengatur operasi mutasi. Probabilitas mutasi menentukan peluang mutasi terjadi atau tidak. *Method* `performMutasi()` yang terdapat pada objek prototipe kromosom yang akan menjalankan proses mutasi.



Gambar 4.25 Diagram Alir Prosedur `mutasi` pada Kelas AGPopulasi

```
public int getSize()
```

Method ini mengembalikan ukuran populasi.

Kembalian:

Kembalian dari *method* ini yaitu ukuran populasi.

```
public ArrayList getKromosom()
```

Method ini mengembalikan sekumpulan objek kromosom yang disimpan dalam *arraylist*.

Kembalian:

Kembalian dari *method* ini yaitu objek-objek kromosom yang terkumpul dalam *arraylist*.

```
public IKromosom<T> getAt(int index)
```

Method ini mengembalikan objek kromosom pada indeks yang telah ditentukan.

Parameter:

indeks posisi/indeks objek kromosom dalam *arraylist*

Kembalian:

Kembalian dari *method* ini yaitu objek kromosom.

```
public void replaceIndividu(int index, IKromosom<T> Kromosom)
```

Method ini mengganti (*replace*) objek kromosom pada indeks yang telah ditentukan dengan objek kromosom baru.

Parameter:

indeks posisi/indeks objek kromosom yang akan digantikan oleh objek kromosom baru

Kromosom objek kromosom baru

Tabel 4.30 Fungsi Anggota Kelas AGPopulasi

4.3.2.12 Kelas AGRandomGenerator

a). **Deskripsi**

Kelas AGRandomGenerator merepresentasikan pembangkit (*generator*) bilangan acak. Bilangan acak yang dibangkitkan antara 0 hingga 1.

b). **Atribut**

Nama	Deskripsi
`static Random randomGenerator`	Objek *generator random* yang disediakan oleh *library mscorlib.dll*.

Tabel 4.31 Atribut Kelas AGRandomGenerator

c). **Fungsi Anggota**

```
public double generateDouble()
```

Method ini membangkitkan bilangan acak dengan tipe data *double* antara 0 hingga 1.

Kembalian:

Kembalian dari *method* ini yaitu bilangan acak *double* antara 0 hingga 1.

```
public float generateFloat()
```

Method ini membangkitkan bilangan acak dengan tipe data *float* antara 0 hingga 1.

Kembalian:

Kembalian dari *method* ini yaitu bilangan acak *float* antara 0 hingga 1.

```
public int generate()
```

Method ini membangkitkan bilangan acak dengan tipe data *integer* antara 0 hingga 1.

Kembalian:

Kembalian dari *method* ini yaitu bilangan acak *integer* antara 0 hingga 2.147.483.647 (nilai maksimum *int* 32-bit).

Tabel 4.32 Fungsi Anggota Kelas AGRandomGenerator

4.3.2.13 *Interface* IRandom

a). **Deskripsi**

IRandom merupakan *interface* untuk pembangkit bilangan acak. Kelas yang mengimplementasikan *interface* ini membangkitkan bilangan acak dengan batas-batas interval (minimum dan atau maksimum) sesuai dengan tipe data masing-masing. Parameter *T* pada kelas ini merupakan tipe data nilai yang dibangkitkan.

b). **Fungsi Anggota**

`T generate()`
Method ini membangkitkan bilangan acak dengan tipe data *T* tanpa ada *range* yang ditentukan. **Kembalian:** Kembalian dari *method* ini yaitu bilangan acak yang dibangkitkan.
`T generate(T max)`
Method ini membangkitkan bilangan acak dengan tipe data *T* dengan nilai maksimum yang telah ditentukan. **Parameter:** *max* nilai maksimum yang dapat dibangkitkan **Kembalian:** Kembalian dari *method* ini yaitu bilangan acak yang dibangkitkan.
`T generate(T min, T max)`
Method ini membangkitkan bilangan acak dengan tipe data *T* dengan *range* nilai yang telah ditentukan. **Parameter:** *min* nilai minimal yang dapat dibangkitkan *max* nilai maksimum yang dapat dibangkitkan **Kembalian:** Kembalian dari *method* ini yaitu bilangan acak yang dibangkitkan.

Tabel 4.33 Fungsi Anggota *Interface* IRandom

4.3.2.14 Kelas AGRandomBool

a). **Deskripsi**

Kelas AGRandomBool membangkitkan nilai acak *boolean*. Kelas ini mengimplementasikan *interface* IRandom. Kelas ini merupakan turunan `IRandom<bool>`.

b). **Atribut**

Nama	Deskripsi
`private AGRandomGenerator generator`	Instansiasi algoritma pembangkit bilangan acak (AGRandomGenerator).

Tabel 4.34 Atribut Kelas AGRandomBool

c). **Fungsi Anggota**

```
public virtual bool generate()
```

Method ini membangkitkan bilangan acak *boolean*.

Kembalian:

Kembalian dari *method* ini yaitu bilangan acak *boolean*.

```
public virtual bool generate(bool max)
```

Method ini membangkitkan bilangan acak *boolean*.

Parameter:

max parameter ini diabaikan

Kembalian:

Kembalian dari *method* ini yaitu bilangan acak *boolean*.

```
public virtual bool generate(bool min, bool max)
```

Method ini membangkitkan bilangan acak *boolean*.

Parameter:

min parameter ini diabaikan

max parameter ini diabaikan

Kembalian:

Kembalian dari *method* ini yaitu bilangan acak *boolean*.

```
public bool generate(double prob)
```

Method ini membangkitkan bilangan *boolean* dengan *prob* sebagai probabilitas *TRUE*.

Parameter:

`prob` probabilitas *TRUE* antara 0 hingga 1

Kembalian:

Kembalian dari *method* ini yaitu bilangan acak *boolean*.

Tabel 4.35 Fungsi Anggota Kelas AGRandomBool

4.3.2.15 Kelas AGRandomDouble

a). **Deskripsi**

Kelas AGRandomDouble membangkitkan nilai acak pecahan dengan tingkat presisi desimal *double*. Kelas ini mengimplementasikan *interface* IRandom. Kelas ini merupakan turunan `IRandom<double>`.

b). **Atribut**

Nama	Deskripsi
`private AGRandomGenerator generator`	Instansiasi algoritma pembangkit bilangan acak (AGRandomGenerator).

Tabel 4.36 Atribut Kelas AGRandomDouble

c). **Fungsi Anggota**

```
public virtual double generate()
```

Method ini membangkitkan nilai acak dengan interval antara 0 hingga 1.

Kembalian:

Kembalian dari *method* ini yaitu nilai acak.

```
public virtual double generate(double max)
```

Method ini membangkitkan nilai acak dengan interval antara 0 hingga *max*.

Parameter:

max nilai maksimum yang dapat dibangkitkan

Kembalian:

Kembalian dari *method* ini yaitu nilai acak.

```
public virtual double generate(double min, double max)
```

Method ini membangkitkan nilai acak dengan interval antara *min* hingga *max*.

Parameter:

min nilai minimum yang dapat dibangkitkan

max nilai maksimum yang dapat dibangkitkan

Kembalian:

Kembalian dari *method* ini yaitu nilai acak.

Tabel 4.37 Fungsi Anggota Kelas AGRandomDouble

4.3.2.16 Kelas AGRandomInteger

a). **Deskripsi**

Kelas AGRandomInteger membangkitkan nilai acak *integer* 32-bit. Kelas ini mengimplementasikan *interface* IRandom. Kelas ini merupakan turunan `IRandom<int>`.

b). **Atribut**

Nama	Deskripsi
`private AGRandomGenerator generator`	Instansiasi algoritma pembangkit bilangan acak (AGRandomGenerator).

Tabel 4.38 Atribut Kelas AGRandomInteger

c). **Fungsi Anggota**

```
public virtual int generate()
```

Method ini membangkitkan nilai acak dengan interval antara 0 hingga 2.147.483.647 (nilai maksimum *int* 32-bit).

Kembalian:

Kembalian dari *method* ini yaitu nilai acak.

```
public virtual int generate(int max)
```

Method ini membangkitkan nilai acak dengan interval antara 0 hingga *max*.

Parameter:

max nilai maksimum yang dapat dibangkitkan

Kembalian:

Kembalian dari *method* ini yaitu nilai acak.

```
public virtual int generate(int min, int max)
```

Method ini membangkitkan nilai acak dengan interval antara *min* hingga *max*.

Parameter:

min nilai minimum yang dapat dibangkitkan

max nilai maksimum yang dapat dibangkitkan

Kembalian:

Kembalian dari *method* ini yaitu nilai acak.

Tabel 4.39 Fungsi Anggota Kelas AGRandomInteger

4.3.2.17 Kelas AGSinglePointCrossover

a). **Deskripsi**

Kelas AGSinglePointCrossover merupakan representasi metode *crossover single point*. Operasi *crossover* ini mengimplementasikan *single point crossover* yaitu dengan memilih sebarang posisi titik *crossover* (*cut-point*) kemudian semua gen (bit) yang berada pada posisi setelah *cut-point* dari kedua kromosom *parent* akan ditukarkan secara silang. Kelas ini mengimplementasikan

interface IOperasiCrossover. Parameter *T* pada kelas ini merupakan tipe data yang digunakan oleh domain kromosom.

b). **Fungsi Anggota**

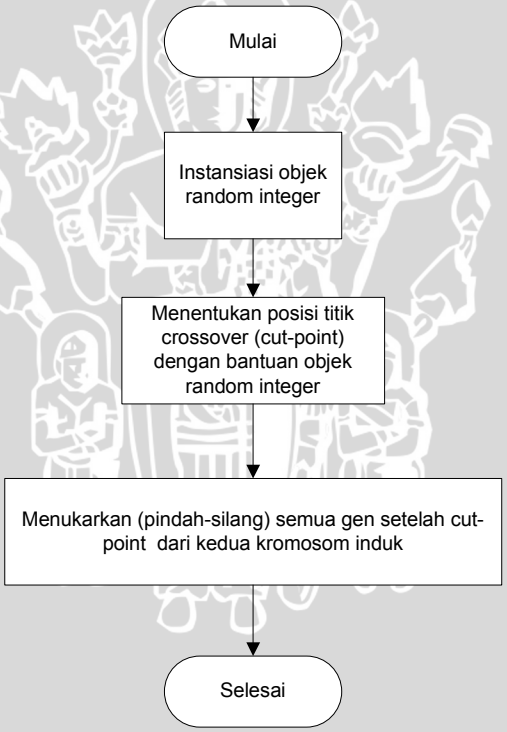
```
public virtual void operateCrossover(ref IKromosom<T> parent1,
ref IKromosom<T> parent2)
```

Method ini menjalankan *crossover single point*. Dalam method ini digunakan objek pembangkit bilangan acak *integer* (AGRandomInteger) untuk menentukan *cut-point* secara acak.

Parameter:

`parent1` *reference variable* objek kromosom induk 1

`parent2` *reference variable* objek kromosom induk 2



Gambar 4.26 Diagram Alir Prosedur `operateCrossover` pada Kelas AGSinglePointCrossover

Tabel 4.40 Fungsi Anggota Kelas AGSinglePointCrossover

4.3.2.18 Kelas AGMutasiRandomUniform

a). **Deskripsi**

Kelas AGMutasiRandomUniform merupakan representasi metode mutasi *random uniform*. Pada mutasi r*andom uniform*, gen yang mengalami mutasi digantikan oleh nilai acak sesuai dengan domain (*lower bound*, *higher bound*). Kelas ini mengimplementasikan *interface* IOperasiMutasi. Parameter *T* pada kelas ini merupakan tipe data yang digunakan oleh domain kromosom.

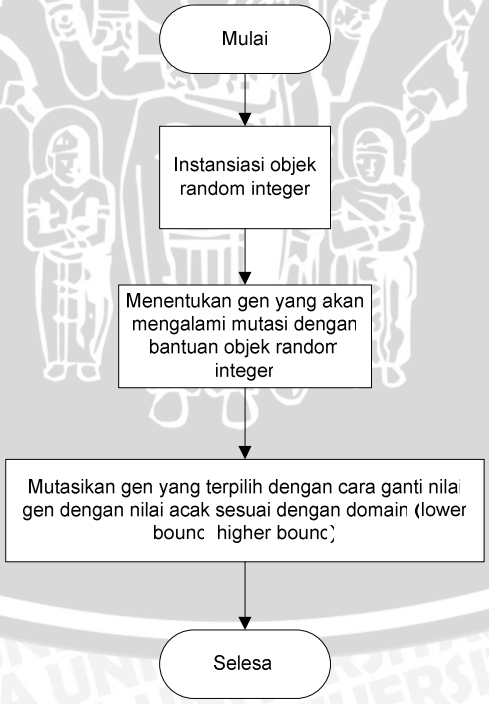
b). **Fungsi Anggota**

```
public virtual void operateMutasi(ref IKromosom<T> Offspring)
```

Method ini menjalankan mutasi *random uniform*. Dalam method ini digunakan objek pembangkit bilangan acak *integer* (AGRandomInteger) untuk menentukan secara acak gen yang akan mengalami mutasi.

Parameter:

`Offspring` *reference variable* objek kromosom yang akan dimutasi



Gambar 4.27 Diagram Alir Prosedur `operateMutasi` pada Kelas AGMutasiRandomUniform

Tabel 4.41 Fungsi Anggota Kelas AGMutasiRandomUniform

4.3.2.19 *Interface* IOperasiSeleksi

a). **Deskripsi**

Interface IOperasiSeleksi merupakan *interface* untuk operasi seleksi. IOperasiSeleksi hanya berisi *signature* fungsi anggota yang digunakan oleh kelas implementasinya. Semua kelas turunan *interface* IOperasiSeleksi harus mengimplementasikan semua fungsi anggota yang dimiliki IOperasiSeleksi. Parameter *T* pada kelas ini merupakan tipe data yang digunakan oleh domain kromosom.

b). **Fungsi Anggota**

```
void performSeleksi(ref ArrayList Kromosom)
```

Method ini menjalankan operasi seleksi.

Parameter:

`Kromosom` *reference variable* kumpulan objek kromosom berupa *arraylist* (populasi) yang akan mengalami mutasi.

Tabel 4.42 Fungsi Anggota *Interface* IOperasiSeleksi

4.3.2.20 Kelas RouletteWheelSelection

a). **Deskripsi**

Kelas RouletteWheelSelection mengimplementasikan metode seleksi roda roulette (*roulette-wheel*). Proses seleksi dimulai dengan memutar roda roulette sebanyak ukuran populasi (*population size*) kali. Pada masing-masing waktu, sebuah kromosom tunggal dipilih untuk sebuah populasi yang baru. Kelas ini mengimplementasikan *interface* IOperasiSeleksi. Parameter *T* pada kelas ini merupakan tipe data yang digunakan oleh domain kromosom.

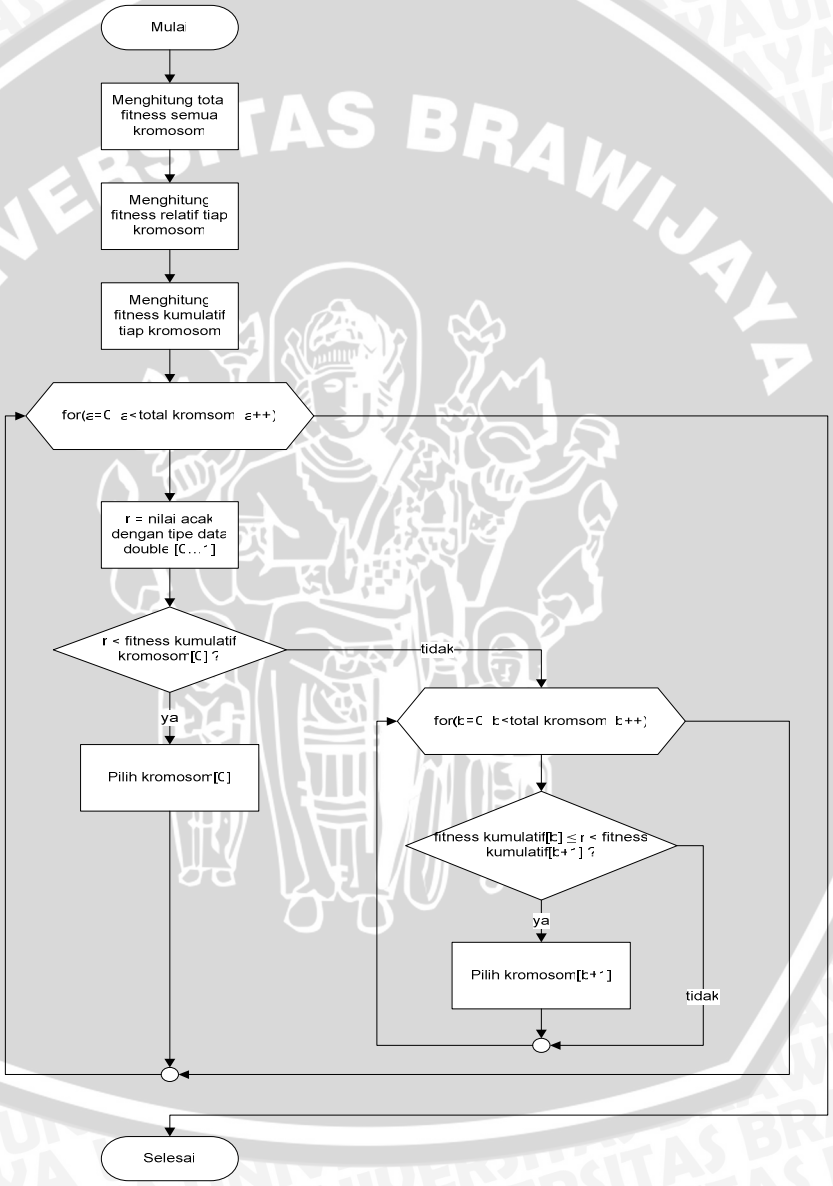
b). **Fungsi Anggota**

```
public virtual void performSeleksi(ref ArrayList kromosom)
```

Method ini menjalankan seleksi roda roulette.

Parameter:

`kromosom` *reference variable arraylist* objek kromosom (populasi) yang akan diseleksi



Gambar 4.28 Diagram Alir Prosedur `performSeleksi` pada Kelas RouletteWheelSelection

Tabel 4.43 Fungsi Anggota Kelas RouletteWheelSelection

4.3.2.21 Kelas AGObserver

a). **Deskripsi**

Kelas AGRandomInteger membangkitkan nilai acak *integer* 32-bit. Kelas ini mengimplementasikan *interface* IRandom. Kelas ini merupakan turunan `IRandom<int>`.

b). **Atribut**

Nama	Deskripsi
`private double fitnessMax = 0`	Nilai *fitness* tertinggi.
`private double fitnessMin = 0`	Nilai *fitness* terendah.
`private double fitnessSum = 0`	Total nilai *fitness*.
`private double fitnessAvg = 0`	Rata-rata nilai *fitness*.
`private IKromosom<T> kromosom_best = null`	Kromosom terbaik yaitu kromosom dengan nilai *fitness* tertinggi.
`private IKromosom<T> kromosom_worst = null`	Kromosom terburuk yaitu kromosom dengan nilai *fitness* terendah.

Tabel 4.44 Atribut Kelas AGObserver

c). **Fungsi Anggota**

```
public AGObserver(AGPopulasi<T> _populasi)
```

Konstruktor ini memanggil *method* `calculateStatistik`.

Parameter:

`_populasi` objek populasi

```
public int findIndexBest(AGPopulasi<T> _populasi)
```

Method ini mencari posisi/indeks kromosom terbaik.

Parameter:

`_populasi` objek populasi

Kembalian:

Kembalian dari *method* ini yaitu posisi/indeks kromosom terbaik.

```
public void findBest(AGPopulasi<T> _populasi)
```

Method ini menyimpan kromosom terbaik setelah diketahui indeks kromosom terbaik oleh *method* `findIndexBest`.

Parameter:

`_populasi` objek populasi

```
public int findIndexWorst(AGPopulasi<T> _populasi)
```

Method ini mencari posisi/indeks kromosom terburuk.

Parameter:

`_populasi` objek populasi

Kembalian:

Kembalian dari *method* ini yaitu posisi/indeks kromosom terburuk.

```
public void findWorst(AGPopulasi<T> _populasi)
```

Method ini menyimpan kromosom terburuk setelah diketahui indeks kromosom terburuk oleh *method* `findIndexWorst`.

Parameter:

`_populasi` objek populasi

```
public void calculateFitnessSumAvg(AGPopulasi<T> _populasi)
```

Method ini menghitung total *fitness* dan rata-rata *fitness* kemudian disimpan ke dalam variabel masing-masing.

Parameter:

`_populasi` objek populasi

```
public void calculateStatistik(AGPopulasi<T> _populasi)
```

Method ini menghitung semua statistik populasi mulai dari *fitness* tertinggi hingga kromosom terburuk.

Parameter:

_populasi objek populasi

```
public void traceOnDebugStatistikFitness(AGPopulasi<T>
_populasi)
```

Method ini menampilkan informasi statistik populasi meliputi statistik *fitness* dan *fitness* masing-masing kromosom. Informasi statistik ditampilkan saat program di*debug*.

Parameter:

_populasi objek populasi

```
public IKromosom<T> getBestKromosom()
```

Method ini mengembalikan objek kromosom terbaik.

Kembalian:

Kembalian dari *method* ini yaitu objek kromosom terbaik.

```
public IKromosom<T> getWorstKromosom()
```

Method ini mengembalikan objek kromosom terburuk.

Kembalian:

Kembalian dari *method* ini yaitu objek kromosom terburuk.

Tabel 4.45 Fungsi Anggota Kelas AGObserver

4.3.2.22 Kelas AGSimple

a). **Deskripsi**

Algoritma Genetika Simpel (*Simple GA*) mengimplementasikan algoritma genetika tanpa *overlapping* populasi yaitu seluruh individu-individu baru menggantikan seluruh individu-individu generasi sebelumnya pada tiap akhir

generasi. Pada *Simple GA* diimplementasikan elitisme dengan tujuan menjaga agar individu terbaik tetap bertahan. Parameter *T* pada kelas ini merupakan tipe data yang digunakan oleh domain kromosom.

b). **Atribut**

Nama	Deskripsi
`private uint maksimumGenerasi`	Batas maksimum generasi.
`private IKromosom<T> BestKromosom`	Kromosom terbaik dari suatu populasi.

Tabel 4.46 Atribut Kelas AGSimple

c). **Fungsi Anggota**

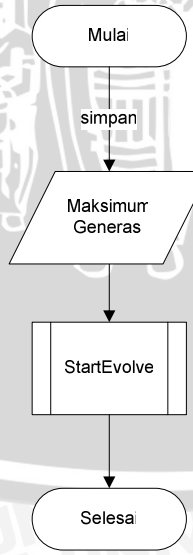
```
public AGSimple(ref AGPopulasi<T> populasi, uint
MaksimumGenerasi)
```

Konstruktor ini menginisialisasi batas maksimum generasi yang diperbolehkan kemudian menjalankan proses evolusi algoritma genetika.

Parameter:

`populasi` *reference variable* objek populasi

`MaksimumGenerasi` batas maksimum generasi



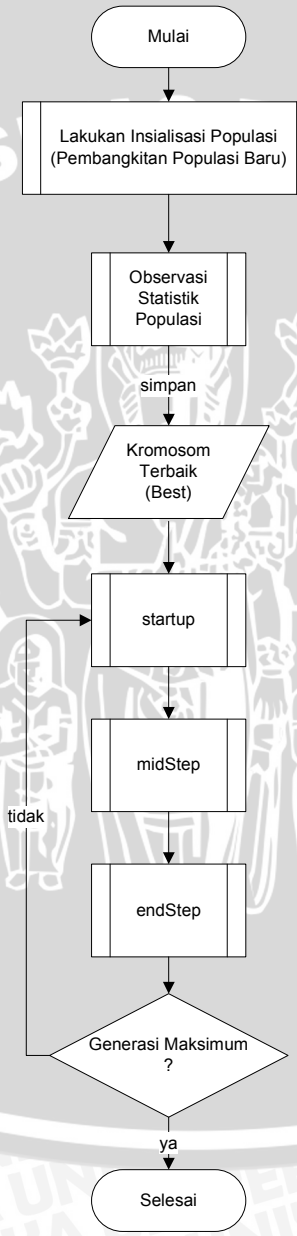
Gambar 4.29 Diagram Alir Konstruktor AGSimple

```
public virtual void StartEvolve(ref AGPopulasi<T> populasi)
```

Method ini menjalankan proses evolusi algoritma genetika mulai dari proses seleksi, *crossover*, mutasi, dan elitisme. Bila telah mencapai batas maksimum generasi, maka proses evolusi dihentikan.

Parameter:

`populasi` *reference variable* populasi



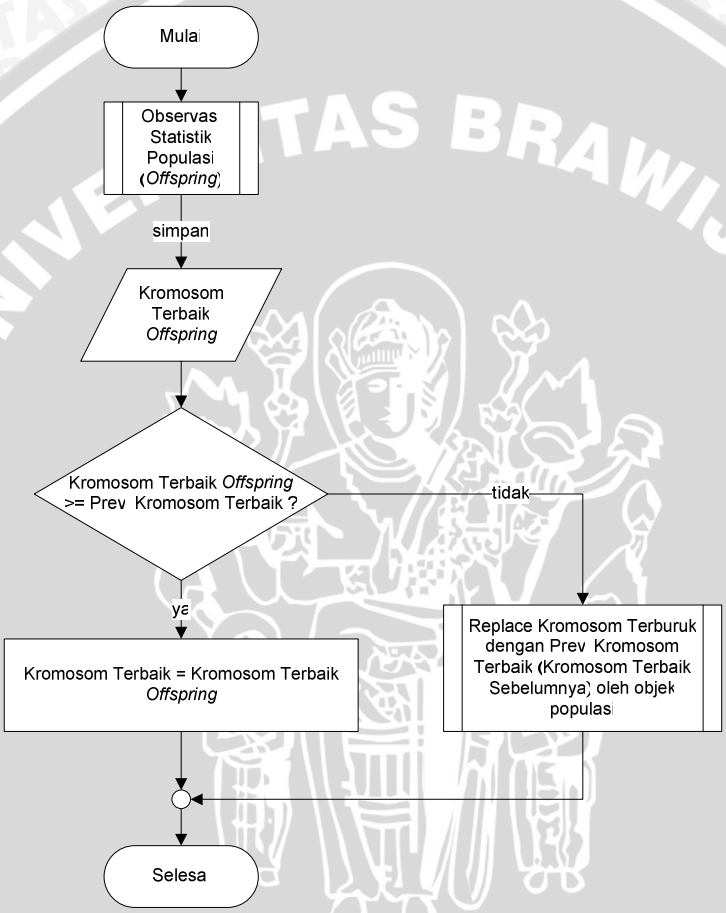
Gambar 4.30 Diagram Alir Prosedur `StartEvolve` pada Kelas AGSimple

```
public void elitist(ref AGPopulasi<T> _populasi)
```

Method ini bertugas menjaga kromosom terbaik agar tetap bertahan.

Parameter:

`populasi` *reference variable* objek populasi



Gambar 4.31 Diagram Alir Prosedur `elitist` pada Kelas AGSimple

```
protected virtual void inisialisasi(ref AGPopulasi<T> populasi)
```

Method ini melakukan pembangkitan populasi baru.

Parameter:

`populasi` *reference variable* objek populasi

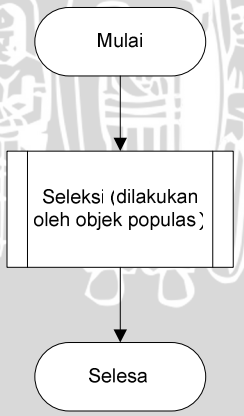
Gambar 4.32 Diagram Alir Prosedur `inisialisasi` pada Kelas AGSimple

```
protected virtual void startup(ref AGPopulasi<T> populasi)
```

Method ini menjalankan proses seleksi kromosom dari suatu populasi pada generasi sekarang. Operasi seleksi yang digunakan disediakan oleh populasi.

Parameter:

`populasi` *reference variable* objek populasi



Gambar 4.33 Diagram Alir Prosedur `startup` pada Kelas AGSimple

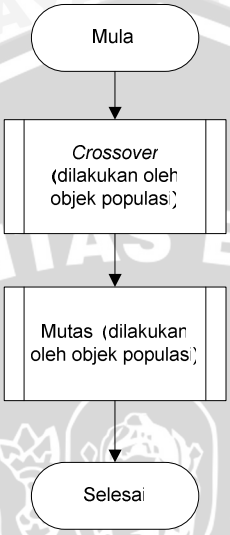
```
protected virtual void midStep(ref AGPopulasi<T> populasi)
```

Method ini menjalankan proses *crossover* dan mutasi sehingga dihasilkan

kromosom *offspring*. Metode *crossover* dan mutasi disediakan oleh populasi.

Parameter:

`populasi` *reference variable* populasi



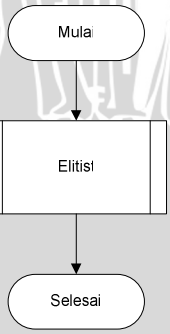
Gambar 4.34 Diagram Alir Prosedur `midStep` pada Kelas AGSimple

```
protected virtual void endStep(ref AGPopulasi<T> populasi)
```

Method ini menjalankan elitisme.

Parameter:

`populasi` *reference variable* objek populasi



Gambar 4.35 Diagram Alir Prosedur `elitist` pada Kelas AGSimple

Tabel 4.47 Fungsi Anggota Kelas AGSimple

BAB V
PENGUJIAN

Rancangan yang telah diimplementasikan memerlukan suatu pengujian. Uji coba dilakukan untuk mengetahui hasil dari penelitian yang dilakukan serta sebagai sarana pemunculan ide-ide untuk pengembangan selanjutnya.

5.1 Validasi *Library*

Pengujian ini bertujuan untuk memastikan apakah *library* AG3NLib berjalan sesuai dengan fungsi yang sudah ditentukan dan benar perhitungannya. Pengujian yang akan dilakukan yaitu dengan membandingkan hasil perhitungan dengan referensi lain yang telah diketahui hasilnya menggunakan data yang sama.

5.1.1 Kasus Uji 1

Pada kasus uji 1, validasi dilakukan dengan membandingkan *library* AG3NLib dengan hasil yang diberikan oleh Fadlisyah, dkk [005].

Diberikan suatu kasus sebagai berikut:

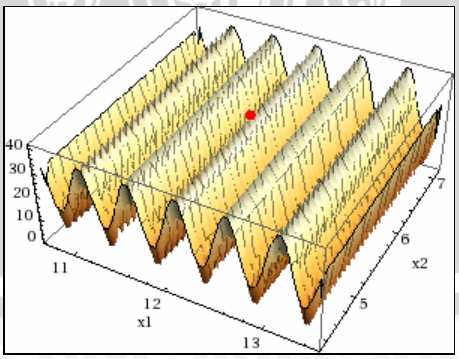
$$\max f(x_1, x_2) = 21{,}5 + x_1 \sin(4\pi x_1) + x_2 \sin(20\pi x_2)$$

dengan:

fungsi kendala: $-3{,}0 \le x_1 \le 12{,}1$

$4{,}1 \le x_2 \le 5{,}8$

Ukuran populasi 10, probabilitas mutasi 1% dan probabilitas crossover 25%.



Gambar 5.1 Grafik Fungsi Kasus Uji 1

Penyelesaian permasalahan tersebut menggunakan AG3NLib dipaparkan secara urut sebagai berikut:

User library harus mendefinisikan persamaan *fitness*.

```
class fungsiFitness : IOperasiFitness<double>
{
   public double evalFitness(IKromosom<double> kromosom)
   {
      double[] val = new double[2];
      AGKromosomRealValue<double> k =
(AGKromosomRealValue<double>) kromosom;
      val = k.getCode();
      return  21.5 + (val[0] * Math.Sin(4 * Math.PI *
val[0])) + (val[1] * Math.Sin(20 * Math.PI * val[1]));
   }
}
```

Kelas fungsiFitness mengimplementasikan *interface* IOperasiFitness dengan cara menurunkan *interface* IOperasiFitness dan mendefinisikan fungsi evaluasi untuk menghitung nilai fitness pada method `evalFitness`.

Kemudian *user library* memberikan parameter kromosom dengan cara mengintansiasi kelas AGParameterKromosom. Parameter dalam konstruktor kelas AGParameterKromosom masing-masing terdiri dari probabilitas mutasi dan probabilitas *crossover*.

```
AGParameterKromosom parameter_kromosom = new
AGParameterKromosom(0.01,0.25);
```

Kelas AGIntervalBounds dibutuhkan untuk menyimpan batasan gen meliputi batas bawah (*lower bound*) dan batas atas (*higher bound*). Batas-batas tersebut menentukan nilai interval gen.

```
AGIntervalBounds<double> interval_x1 = new
AGIntervalBounds<double>(-3.0, 12.1);

AGIntervalBounds<double> interval_x2 = new
AGIntervalBounds<double>(4.1, 5.8);
```

Kelas AGIntervalSet digunakan untuk menyimpan domain x_1 dan x_2.

```
AGIntervalSet<double>[] setNilai = new
AGIntervalSet<double>[2];
```

```
        setNilai[0] = new AGIntervalSet<double>(interval_x1, new
AGRandomDouble());

        setNilai[1] = new AGIntervalSet<double>(interval_x2, new
AGRandomDouble());
```

Langkah berikutnya yaitu instansiasi objek konfigurasi kromosom. Konfigurasi kromosom berisi objek parameter kromosom, objek yang mendefinisikan domain variabel x_1 dan x_2, objek fungsi *fitness* serta objek operasi *crossover* dan operasi mutasi yang akan digunakan.

Sebelum dilakukan instansiasi objek konfigurasi kromosom, terlebih dulu dilakukan instansiasi objek kelas fungsiFitness.

```
        fungsiFitness fungsiObjektif = new fungsiFitness();

        AGBlokKonfigurasiKromosom<double> konfigurasiKromosom =
new AGBlokKonfigurasiKromosom<double>(setNilai, 2,
parameter_kromosom, new AGSinglePointCrossover<double>(), new
AGMutasiRandomUniform<double>(), fungsiObjektif);
```

Setelah didefinisikan konfigurasi kromosom, *user library* selanjutnya dapat menginstansiasi objek kromosom prototipe.

```
        AGKromosomRealValue<double> prototipe = new
AGKromosomRealValue<double>(konfigurasiKromosom,2);
```

Setelah itu, objek populasi siap dibuat. Parameter pertama merupakan kromosom prototipe, parameter kedua yaitu operasi seleksi yang digunakan dan parameter ketiga yaitu ukuran populasi.

```
        AGPopulasi<double> populasi = new
AGPopulasi<double>(prototipe, new
RouletteWheelSelection<double>(),10);
```

Kemudian algoritma genetika simpel siap dijalankan.

```
        AGSimple<double> algoritma = new AGSimple<double>(ref
populasi, 1000);
```

Untuk mengetahui statistik populasi (misalnya kromosom terbaik) setelah proses evolusi algoritma genetika dijalankan, digunakan kelas AGObserver.

```
AGObserver<double> observer = new
AGObserver<double>(populasi);

AGKromosomRealValue<double> bestChromosome =
(AGKromosomRealValue<double>)observer.getBestKromosom();
```

Fenotipe gen (pada contoh kasus ini genotip sama dengan fenotip) dan nilai *fitness* kromosom terbaik dapat ditampilkan pada *console window*/*command-line interface*.

```
Console.WriteLine("Best Kromosom: x=
"+bestChromosome.getAt(0)+"; y= "+bestChromosome.getAt(1)+"
nilai fitness = "+bestChromosome.getFitness());
```

Pada validasi ini, implementasi *library* AG3NLib dengan kasus tersebut dijalankan sebanyak 9 kali dengan parameter jumlah generasi yang berbeda yaitu 10.000 generasi, 50.000 generasi, dan 100.000 generasi. Nilai *fitness* kromosom terbaik yang dihasilkan kemudian dibandingkan dengan hasil yang diberikan Fadlisyah, dkk [005]. Tabel 5.1 menunjukkan nilai *fitness* dan nilai variabel (genotip) kromosom terbaik (*best chromosome*) dengan generasi maksimum yang berbeda-beda.

Tabel 5.1 Hasil Implementasi AG3NLib pada Kasus Uji 1

Jumlah Generasi	Pengujian ke-	Nilai Variabel Kromosom Terbaik		Nilai *Fitness* Kromosom Terbaik
		x_1	x_2	
10.000	1	12,0877	5,7247	38,0096
	2	11,6039	5,7253	38,4243
	3	11,6424	5,6239	38,4756
50.000	1	11,6253	5,7254	38,8491
	2	11,6256	5,7249	38,8501

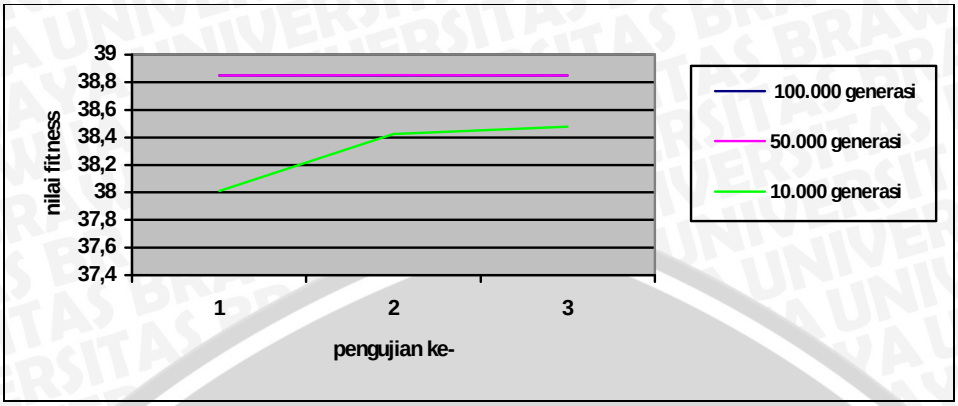
Jumlah Generasi	Pengujian ke-	Nilai Variabel Kromosom Terbaik		Nilai *Fitness* Kromosom Terbaik
		x_1	x_2	
50.000	3	11,6261	5,7247	38,8490
100.000	1	11,6241	5,7250	38,8485
	2	11,6241	5,7248	38,8476
	3	11,6239	5,7251	38,8479

Perbandingan nilai *fitness* kromosom terbaik hasil perhitungan dengan jumlah generasi 100.000 generasi pada kasus optimisasi tersebut di atas menggunakan AG3NLib dengan hasil yang diberikan Fadlisyah, dkk [005] ditunjukkan oleh tabel 5.2.

Tabel 5.2 Perbandingan Nilai *Fitness* Kromosom Terbaik Hasil Perhitungan AG3NLib dengan Fadlisyah, dkk [005]

No	Pengujian dengan	Nilai Fitness Kromosom Terbaik	Presentase Perbandingan dengan (1)
1	Fadlisyah, dkk [005]	38,818208	-
2	AG3NLib	38,8485	3,03

Grafik pada gambar 5.2 menunjukkan perubahan nilai *fitness* kromosom terbaik pada kasus uji 1 dengan jumlah generasi yang berbeda-beda.



Gambar 5.2 Grafik Perubahan Nilai *Fitness* Kromosom Terbaik pada Kasus Uji 1 dan Perbandingannya

Kesimpulan dari validasi *library* menunjukkan bahwa *library* AG3NLib berjalan sesuai dengan fungsi yang telah ditentukan dan benar perhitungannya. Berdasarkan gambar 5.2, hasil yang didapatkan akan konsisten ketika parameter jumlah generasi makin besar yaitu dimulai dari 50.000 generasi.

5.1.2 Kasus Uji 2

Pada kasus uji 2, validasi bertujuan untuk membuktikan *library* AG3NLib memberikan hasil perhitungan yang benar dibandingkan dengan perhitungan menggunakan metode konvensional.

Diberikan suatu kasus sebagai berikut:

$$\max f(x_1, x_2) = 2x_1 - x_2$$

dengan:

fungsi kendala: $0 \le x_1 \le 5$

$$2 \le x_2 \le 7$$

Kasus uji di atas diselesaikan secara matematis, menghasilkan solusi maksimum untuk nilai x_1 sebesar mungkin dan nilai x_2 sekecil mungkin. Dengan batasan fungsi kendala tersebut di atas, maka solusi didapatkan pada $x_1 = 5$ dan $x_2 = 2$ sehingga f(5,2) = 8.

Langkah-langkah penyelesaian masalah tersebut menggunakan *library* AG3NLib secara garis besar sama dengan penyelesaian kasus uji sebelumnya.

Perbedaan terletak pada fungsi objektif (fungsi *fitness*) dan *domain* variabel solusi.

Nilai yang diperoleh dalam pengujian untuk kasus uji 2

x_1 = 4,985396

x_2 = 2,032763

fitness kromosom terbaik atau f(4,985396, 2,032763) = 7,938028

Selisih antara hasil perhitungan secara matematis dengan perhitungan AG3NLib untuk nilai fitness terbaik yaitu:

$$\frac{8-7{,}938028}{8}\times 100\% = 0{,}77\%$$

Kesimpulan dari validasi *library* dengan kasus uji 2 menunjukkan bahwa *library* AG3NLib memberikan hasil perhitungan yang benar. Perbedaan dapat terjadi karena algoritma genetika bersifat stokastik.

5.2 Pengujian *Execution Time*

Pengujian ini bertujuan untuk mengetahui waktu eksekusi yang dibutuhkan saat proses evolusi algoritma genetika pada *library* AG3NLib dijalankan. Pengujian ini dilakukan dari sisi program *user library* yang menggunakan AG3NLib pada kasus uji 1. Waktu eksekusi didapatkan dengan menghitung waktu sesaat ketika proses evolusi akan dijalankan dan waktu sesaat sesudah proses evolusi dijalankan (kelas AGSimple diinstansiasi) kemudian dihitung intervalnya.

```
DateTime start = DateTime.Now;

AGSimple<double> algoritma = new AGSimple<double>(ref populasi,
1000);

DateTime end = DateTime.Now;
TimeSpan duration = end - start;

Console.WriteLine(string.Format("Hours{0} Minutes{1} Seconds {2},
Milliseconds {3}", duration.Hours, duration.Minutes,
duration.Seconds, duration.Milliseconds));
```

Untuk mendapatkan waktu sesaat akan dijalankan dan sesudah proses evolusi dijalankan digunakan *property* `DateTime.Now` yang terdapat pada kelas DateTime pada *library* mscorlib.dll. `DateTime.Now` digunakan untuk

mendapatkan tanggal dan waktu lokal (*local time*) saat ini pada komputer yang digunakan. Format keluaran `DateTime.Now` yaitu "month/date/year hh:mm:ss". Interval waktu sesaat sesudah dan sebelum proses evolusi dijalankan diperoleh dengan menghitung selisihnya. Untuk memudahkan mengetahui detil interval waktu tersebut yaitu jam, menit, detik, milidetik, dan sebagainya, maka interval waktu tersebut disimpan ke dalam objek TimeSpan. TimeSpan merupakan kelas yang disediakan oleh *library* mscorlib.dll yang merepresentasikan interval waktu.

Berdasarkan cara kerja dari *library* algoritma genetika yang dirancang, dapat disimpulkan bahwa faktor yang paling mempengaruhi waktu eksekusi pada saat proses evolusi algoritma genetika dijalankan untuk satu kasus yang sama adalah parameter ukuran populasi, jumlah generasi, probabilitas *crossover*, dan probabilitas mutasi.

Parameter yang akan digunakan untuk pengujian ini ditunjukkan pada tabel 5.3 berikut ini.

Tabel 5.3 Nilai Parameter yang Digunakan pada Pengujian *Execution Time*

Pengujian ke-	Parameter			
	Ukuran Populasi	Jumlah Generasi	Prob. Crossover	Prob. Mutasi
1	10	10000	25%	1%
2	20	10000	25%	1%
3	50	10000	25%	1%
4	10	50000	25%	1%
5	10	100000	25%	1%
6	10	10000	50%	1%
7	10	10000	75%	1%
8	10	10000	25%	10%
9	10	10000	25%	50%

Pengujian dilakukan dengan mengubah-ubah nilai parameter-parameter tersebut. Nilai-nilai dari parameter yang sedang tidak diuji akan menggunakan nilai pada pengujian pertama sehingga dapat terlihat perbedaan yang diakibatkan dari perubahan tiap-tiap nilai parameter. Pengujian ini menggunakan perangkat keras komputer dan perangkat lunak dengan spesifikasi yang tunjukkan pada tabel 5.4.

Tabel 5.4 Spesifikasi Perangkat Keras Komputer dan Perangkat Lunak Pada Pengujian Waktu Eksekusi

Prosesor	Intel Core 2 Duo T6500 2,10 GHz
Memori (RAM)	2,00 GB
Sistem Operasi	Windows Vista Home Premium Service Pack 2

Hasil dari pengujian pada masing-masing perubahan nilai parameter ditunjukkan pada tabel 5.5.

Tabel 5.5 Hasil Pengujian *Execution Time*

Pengujian ke-	Waktu eksekusi (dalam milidetik)
1	233
2	508
3	1357
4	1429
5	2227
6	272
7	428
8	287
9	368

Dari pengujian waktu eksekusi didapatkan kesimpulan bahwa semakin besar ukuran populasi dan jumlah generasi, maka waktu eksekusi akan semakin bertambah. Sedangkan untuk nilai probabilitas *crossover* dan mutasi yang semakin besar akan memberikan kemungkinan penambahan waktu eksekusi karena peluang untuk proses *crossover* dan mutasi dijalankan semakin besar pula.

BAB VI
PENUTUP

6.1 Kesimpulan

Berdasarkan hasil perancangan, implementasi dan pengujian yang dilakukan, maka diambil kesimpulan sebagai berikut:

1. *Library* algoritma genetika terdiri dari kelas-kelas dan *interface-interface* yang dikelompokkan dalam *namespace* Algoritma, General, Kromosom, Kromosom.OperasiCrossover, Kromosom.OperasiMutasi, Kromosom.Representasi, Populasi, Populasi.OperasiSeleksi, Statistik.Observer sesuai dengan desain algoritma genetika.
2. Pengujian validasi *library* yaitu membandingkan hasil perhitungan AG3NLib dengan hasil perhitungan Fadlisyah, dkk[005] dan metode konvensional pada masing-masing kasus uji didapat perbedaan *fitness* sebesar 3,03% antara AG3NLib dengan Fadlisyah, dkk[005] pada kasus uji 1 sedangkan pada kasus uji 2 terdapat perbedaan *fitness* sebesar 0,77% antara AG3NLib dengan metode konvensional. Hal ini menunjukkan bahwa AG3NLib berjalan sesuai dengan fungsi yang telah ditentukan dan benar perhitungannya.
3. Pengujian validasi pada kasus uji 1 diperoleh bahwa hasil perhitungan dengan algoritma genetika akan konsisten bila parameter jumlah generasi makin besar.
4. Performa waktu eksekusi dari algoritma, yang ditunjukkan pada pengujian program yang menggunakan library AG3NLib, dipengaruhi secara signifikan oleh parameter-parameter yaitu jumlah populasi dan jumlah generasi. Semakin besar nilai parameternya, proses eksekusi akan berlangsung lebih lama.

6.2 Saran

Saran yang dapat diberikan untuk pengembangan *library* ini antara lain:

1. *Library* ini dapat dikembangkan fitur-fiturnya yaitu representasi kromosom dengan menurunkan (*inherit*) kelas AGKromosomRealValue, serta metode *crossover*, metode mutasi, dan metode seleksi dengan cara

membuat kelas baru yang mengimplementasikan *interface* IOperasiCrossover untuk metode *crossover*, IOperasiMutasi untuk metode mutasi, dan IOperasiSeleksi untuk metode seleksi.

2. Algoritma yang digunakan oleh AG3NLib yaitu simple GA. *Library* ini selanjutnya dapat dikembangkan untuk incremental GA, paralel GA, dan sebagainya dengan menurunkan kelas AGSimple.
3. Dalam AG3NLib, populasi berisi *araylist* kromosom-kromosom. Selanjutnya dapat dikembangkan untuk struktur data yang lain, misalnya *tree* dan *list.*

DAFTAR PUSTAKA

[001] Anonymous. 2010. *Algoritma Genetika.* http://lecturer.eepisits.edu/~kangedi/materikuliah/Kecerdasan Buatan/Bab 7 Algoritma Genetika.pdf, diakses tanggal 22 Juli 2010.

[002] Aribowo, A., Lukas, S. & Gunawan, Martin. 2008. Penerapan Algoritma Genetika pada Penentuan Komposisi Pakan Ayam Petelur. *Seminar Nasional Aplikasi Teknologi Informasi 2008*, (Online), (http://journal.uii.ac.id/index.php/Snati/article/view/384/299, *diakses 3 Agustus 2010*).

[003] Basuki, Achmad. 2003**.** *Algoritma Genetika Suatu Alternatif Penyelesaian Permasalahan Searching, Optimasi dan Machine Learning***.** Surabaya **:** PENS-ITS.

[004] Chambers, Lance. 2001. *The Practical Handbook of Genetic Algorithms Applications, Second Edition.* Boca Raton: Chapman & Hall / CRC.

[005] Fadliansyah, Arnawan, Faisal. 2009. *Algoritma Genetik.* Yogyakarta: Graha Ilmu.

[006] Fowler, Martin et al. 2002. *Patterns of Enterprise Application Architecture*. Boston: Addison Wesley.

[007] Horstmann, Cay. 2006. *Object-Oriented Design & Patterns, Second Edition*. New Jersey: John Wiley & Sons, Inc.

[008] Knoernschild, Kirk. 2001. *Java Design : Objects, UML, and Process*. New Jersey: Addison Wesley.

[009] Microsoft Corp. 2011. *About Dynamic-Link Libraries* http://msdn.microsoft.com/en-us/library/windows/desktop/ms681914(v=vs.85).aspx, diakses tanggal 10 Desember 2011.

[010] Mitchell, Melanie. 1996. *An Inroduction to Genetic Algorithms.* London: MIT Press.

[011] Obitko, Marek. 1998. *Introduction to Genetic Algorithms.* http://www.obitko.com/tutorials/genetic-algorithms/, diakses tanggal 22 Juli 2010.

[012] Pressman, Roger S. 2001. *Software Engineering : A Practitioner's Approach, 5th Edition.* New York: McGrawHill.

[013] Rumbaugh, J., Jacobson, I., Booch, Grady. 2004. *The Unified Modelling Language Reference Manual, Second Edition.* Boston: Addison-Wesley.

[014] Sinaga, Edison. 2009. *Implementasi Algoritma Genetika Dalam Penyusunan Teka Teki Silang.* Skripsi. Program Studi Ilmu Komputer, Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Sumatera Utara, Medan.

[015] Sivanandam, S.N dan Deepa, S.N. 2008. *Introduction to Genetic Algorithms.* New York : Springer-Verlag Berlin Heidelberg.

[016] Sommerville, Ian. 2004. *Software Engineering, Eighth Edition.* Harlow: Addison-Wesley.

[017] Utley, Craig. 2002. *Why you should move to C#.* http://www.techrepublic.com/article/why-you-should-move-to-c/1050356, diakses tanggal 20 Desember 2011.