BAB 2
TINJAUAN PUSTAKA

Tinjauan pustaka merupakan penjelasan berbagai konsep dasar dari teori - teori yang berkaitan dengan implementasi algoritma genetika untuk menyelesaikan permainan *puzzle* kakuro.

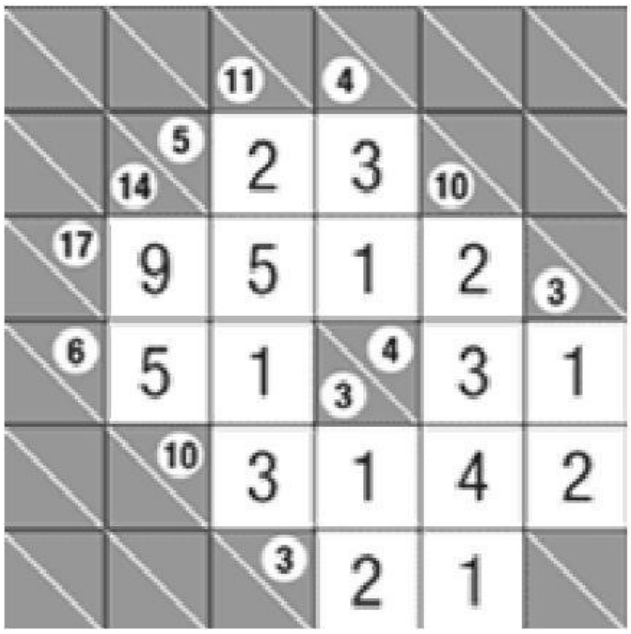
2.1 Pengertian Permainan

Game atau permainan merupakan media hiburan yang sudah dikenal sejak lama. *Game* dapat dimainkan berbagai umur tua maupun muda. *Game* juga sudah berkembang dengan pesat sesuai dengan perkembangan teknologi, dari *game* sederhana sampai *game* modern saat ini. Hal ini terbukti dengan adanya perkembangan jenis, produk, serta alat yang digunakan.

2.2 Permainan *Puzzle* Kakuro

Kakuro adalah sebuah permainan *puzzle* yang bermula bernama *cross sums. Puzzle,* pertama dikeluarkan pada tahun 1966 oleh *Dell Megazine* di Amerika Serikat. Sepuluh tahun kemudian, *Dell Megazine* memperkenalkan *puzzle* sudoku pada dunia. Maki Kaji presiden dari *Nicoli Puzzle,* pada tahun 1980 membawa masuk *cross games* ke Jepang. Maki Kaji memberi nama *puzzle* buatannya adalah kasan kurosu (penjumlahan silang). Pada tahun 1986, *Nicoli* memberi nama dagang *game* dengan nama kakuro (kependekan dari kasan kurosu). September 2005 adalah titik penentunya, dimana *game puzzle* kakuro diperkenalkan ke barat oleh *The Guardian* dan *The Daily Mail* yang menerbitkan *puzzle kakuro* setiap harinya di Inggris dan setelah itu *game puzzle* kakuro ini menyebar keseluruh dunia.

Permainan ini adalah permainan penjumlahan angka dengan aturan mainnya yaitu dengan mengisi setiap kotak yang tersedia dengan beberapa syarat dalam pengisiannya. Soal-soal dalam permainan *puzzle* kakuro bersifat unik, karena soal-soal ini akan saling terkait baik dengan soal lainnya maupun jawaban dalam menyelesaikan permainan *puzzle* kakuro. Berikut adalah contoh dari permainan *puzzle* kakuro pada gambar 2.1 :

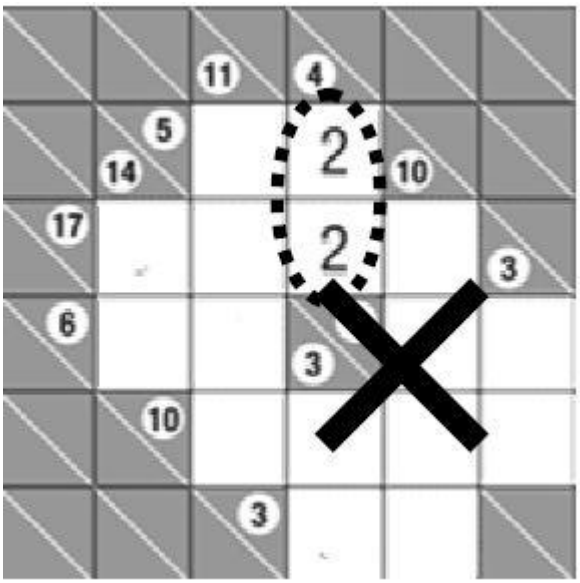


Gambar 2. 1 Contoh *Puzzle* Kakuro Yang Memenuhi Syarat Permainan

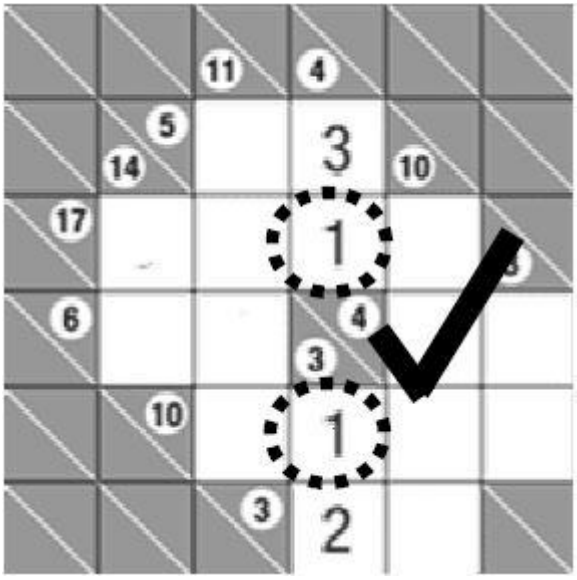
Adapun syarat – syarat dalam pengisian kotak jawaban pada permainan kakuro yaitu sebagai berikut : **[**1**]**

1. Setiap kotak hanya boleh diisikan dengan bilangan bulat dari 1 sampai 9.
2. Setiap kotak yang berurutan dalam satu lajur (lajur yang dimaksud adalah deretan kotak yang berurutan dalam satu baris atau kolom) tidak boleh memiliki angka yang sama.
3. Isi dari setiap lajur harus memiliki jumlah yang sama pada ujung kiri (untuk baris) atau ujung atas (untuk kolom) lajur tersebut.

Berikut adalah contoh dari *puzzle* kakuro yang tidak sesuai dengan peraturan terdapat pada gambar 2.2 dan yang sesuai dengan peraturan terdapat pada gambar 2.3 :



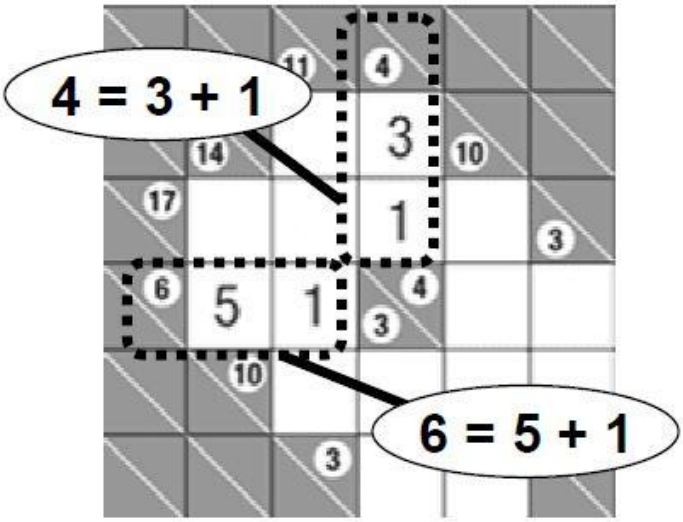
Gambar 2.2 Contoh *Puzzle* Kakuro Yang Tidak Sesuai Aturan Permainan



Gambar 2.3 Contoh *Puzzle* Kakuro Yang Sesuai Aturan Permainan

Pengisian jawaban yang ditunjukan pada gambar 2.2 merupakan contoh yang salah karena terdapat dua angka atau lebih pada satu lajur. Sedangkan pada gambar 2.3 merupakan contoh yang benar, meskipun terdapat angka yang sama pada satu kolom tetapi angka tersebut terdapat pada lajur yang berbeda sehingga

tidak menyalahi aturan dari permainan *puzzle* kakuro. Sedangkan untuk pengisian jawaban yang benar dapat dilihat pada gambar 2.4 :



Gambar 2.4 Contoh Pengisian Jawaban

2.3 Pengertian Algoritma

Algoritma merupakan fondasi yang harus dikuasai untuk menyelesaikan suatu masalah secara terstruktur, efektif dan efisien. Algoritma dapat didefinisikan sebagai suatu prosedur yang jelas untuk menyelesaikan suatu persoalan dengan menggunakan langkah – langkah tertentu dan terbatas jumlahnya [**6**].

2.3.1 Ciri Algoritma

Ada beberapa ciri dari algoritma, beberapa diantaranya adalah sebagai berikut [**6**]:

1. Algoritma mempunyai awal dan akhir, suatu algoritma harus berhenti setelah mengerjakan serangkaian tugas. Dengan kata lain, suatu algoritma memiliki langkah yang terbatas.
2. Setiap langkah harus didefinisikan dengan tepat sehingga tidak memiliki arti ganda, tidak membingungkan.
3. Memiliki masukan atau kondisi awal.
4. Memiliki keluaran atau kondisi akhir.
5. Algoritma harus efektif, bila diikuti benar – benar maka akan menyelesaikan masalah.

2.3.2 Sifat algoritma

Berdasarkan ciri dan definisi dari algoritma yang telah dipaparkan sebelumnya, dapat disimpulkan bahwa sifat utama suatu algoritma adalah sebagai berikut [**6**]:

1. ***Input*** : Suatu algoritma memiliki input atau kondisi awal sebelum dilaksanakan, bisa berupa nilai – nilai peubah yang diambil dari himpunan khusus.
2. ***Output*** : Suatu algoritma akan menghasilkan output setelah dilaksanakan atau algoritma akan mengubah kondisi awal menjadi kondisi akhir, di mana nilai output diperoleh dari nilai input yang telah diproses melalui algoritma.
3. ***Definiteness*** : Langkah – langkah yang dituliskan dalam algoritma terdefinisi dengan jelas sehingga mudah dilaksanakan oleh pengguna algoritma.
4. ***Finitness*** : Suat ualgoritma harus memberi kondisi akhir atau output setelah sejumlah langkah yang terbatas jumlahnya dilakukan terhadap kondisi awal atau input yang diberikan.
5. ***Efektiveness*** : Setiap langkah dalam algoritma bisa dilaksanakan dalam suatu selang waktu tertentu sehingga pada akhirnya didapatkan solusi sesuai yang diharapkan.
6. ***Generality*** : Langkah – langkah algoritma berlaku untuk setiap himpunan input yang sesuai dengan persoalan yang diberikan, tidak hanya untuk himpunan tertentu.

2.4 Algoritma Genetika

Algoritma genetika merupakan suatu metode heuristic yang dikembangkan berdasarkan prinsip genetika dan proses seleksi alamiah Teori Evolusi Darwin. Metode optimasi dikembangkan oleh John Holland sekitar tahun 1960-an dan dipopulerkan oleh salah seorang mahasiswanya, David Goldreg pada tahun 1980-an. Proses pencarian penyelesaian atau proses terpilihnya suatu individu untuk bertahan hidup dalam proses evolusi.

Dalam teori evolusi Darwin, suatu individu tercipta secara acak kemudian berkembang biak melalui proses reproduksi sehingga terbentuk sekumpulan

individu sebagai populasi. Setiap individu dalam populasi mempunyai tingkat kebugaran yang berbeda-beda. Tingkat kebugaran ini menentukan seberapa kuat untuk tetap betahan hidup dalam populasinya. Sebagian individu tetap bertahan hidup dan sebagian lainnya mati.

Pada dasarnya, algoritma genetika merupakan metode pencarian yang didasarkan pada proses evolusi alamiah, yaitu terbentuknya populasi awal secara acak yang terdiri dari individu-individu dengan sifat yang bergantung pada gen-gen dalam kromosomnya. Individu-individu melakukan proses reproduksi untuk melahirkan keturunan. Sifat keturunan dibentuk dari kombinasi sifat kedua induknya atau mewarisi sifat-sifat induknya [**2**].

2.4.1 Definisi Penting Dalam Algoritma Genetika

Dalam algoritma genetika terdapat elemen-elemen yang digunakan sebagai representas sebuah variable, elemen tersebut saling berkaitan serta mempunyai peranan masing-masing , berikut adalah penjelesan elemen-elemen tersebut :

1. *Allele*

   Allele merupakan suatu nilai yang terdapat pada suatu gen. Satu gen mempunyai satu nilai yang disebut *allele*. *Allele* dapat direpresentasikan dalam notasi yang telah ditentukan (biner , float, integer, maupun karakter).

2. Gen

   Gen merupakan nilai yang menyatakan satuan dasar yang membentuk suatu arti tertentu dalam satu kesatuan gen yang dinamakan kromosom. Gen mewakili satu atau lebih variabel sebagai solusi yang ingin dicapai. Gen dapat direpresentasikan dalam bentuk berikut :

   String bit : 10011, 01101, 11101, dst.
   Bilangan real : 65.65, -67.98, 592.88, dst.
   Elemen permutasi : E2, E10, E6, dst.
   Daftar aturan : R1, R2, R3, dst.
   Elemen program : pemograman genetika.

3. Kromosom atau Individu

   Kromosom atau individu merupakan gabungan dari kumpulan gen-gen yang membentuk nilai tertentu dan menyatakan solusi yang mungkin dari suatu permasalahan.

4. Populasi

   Populasi merupakan sekumpulan individu yang akan diproses bersama dalam satu satuan siklus evolusi.

5. *Fitness* (Nilai *fitness*)

   Fitness merupakan suatu nilai yang menyatakan seberapa baik suatu individu yang didapatkan. Semakin besar nilai *fitness* semakin baik suatu individu untuk bertahan hidup.

6. Seleksi

   Seleksi merupakan proses pemilihan dua individu terbaik/ memiliki nilai *fitness* yang tinggi yang akan dijadikan sebagai induk (*parent*) untuk digunakan dalam proses regenerasi. Seleksi juga berfungsi memberikan kesempatan reproduksi yang lebih besar bagi anggota populasi yang paling fit.

7. *Crossover*

   Crossover merupakan proses pertukaran atau kawin silang gen-gen dari dua induk tertentu.

8. Mutasi

   Mutasi merupakan proses pergantian salah satu gen yang terpilih dengan nilai tertentu.

9. Generasi

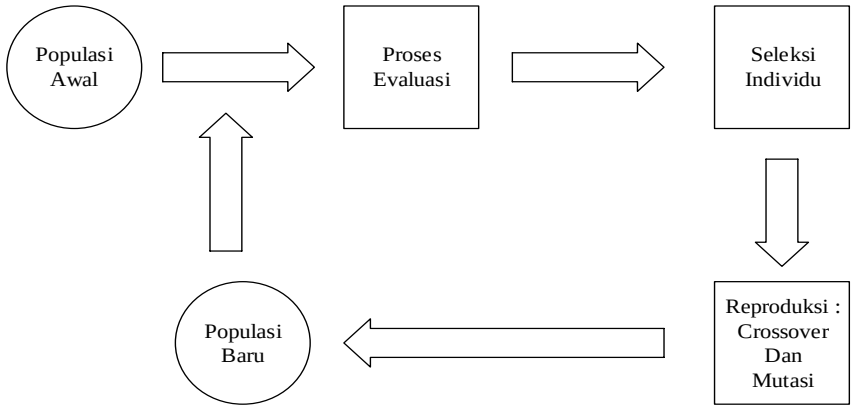
   Generasi merupakan urutan iterasi dalam satu kesatuan proses genetika, dimana beberapa kromososm terbaik dari generasi sebelumnya bergabung dan membuat kromosom baru dengan solusi yang lebih baik.

10. *Offspring*

    Offspring merupakan kromosom baru yang dihasilkan setelah melewati suatu generasi. Dalam hal ini *offspring* tersebut merupakan hasil kromosom yang dibangkitkan setelah melakukan proses seleksi.

2.4.2 Siklus Algoritma Genetika

Siklus algoritma genetika dibutuhkan pada setiap kasus, siklus petama kali memperkenalkan siklus adalah oleh David Goldberg. Siklus dimulai dari membuat populasi awal secara acak, kemudian setiap individu dihitung nilai *fitness*-nya. Proses berikut adalah menyeleksi individu terbaik, kemudian dilakukan *crossover* dan dilanjutkan oleh proses mutasi sehingga terbentuk populasi baru. Selanjutnya populasi baru ini mengalami siklus yang sama dengan populasi sebelumnya. Proses tersebut berlangsung hingga generasi ke –n. Berikut siklus algoritma genetika menurut David Goldberg pada gambar 2.1 [**7**]:



Gambar 2.5 Siklus Algoritma Genetika David Goldberg

2.4.3 Komponen Utama Algoritma Genetika

Komponen-komponen utama algoritma genetika saling berketergantungan dan berkaitan satu sama lain, berikut penjelasan dari komponen tersebut [**8**]:

2.4.3.1 Teknik Pengkodean

Teknik pengkodean merupakan bagian penting dalam algortima genetika. Proses ini diperlukan dalam kaitan dengan peranan kromosom sebagai representasi penyelesaian masalah. Kromosom gabungan dari gen-gen yang membentuk nilai tertentu. Satu gen akan mewakili satu *variable*, agar dapat diproses melalui algoritma genetika. Gen dapat direpresentasikan dalam bentuk :

string bit, bilangan real, daftar aturan, elemen permutasi, elemen program, atau representasi lainnya yang dapat diimplementasikan untuk operator genetika.

2.4.3.2 Membangkitkan Populasi Awal

Membangkitkan populasi awal adalah proses membangkitkan sejumlah individu secara acak atau melalu *procedure* tertentu. Ukuran untuk populasi tergantung pada masalah yang akan diselesaikan dan jenis operator genetika yang diimplementasikan. Setelah ukuran populasi ditentukan, kemudian dilakukan pembangkitan populasi awal.

Teknik dalam pembangkitan populasi awal ini ada beberapa cara, diantaranya adalah sebagai berikut :

1. *Random search*

   Pencarian solusi dimulai dari suatu titik uji tertentu. Titik uji tersebut dianggap sebagai alternatif solusi yang disebut sebagai populasi.
2. *Random Generator*

   Random generator melibatkan pembangkitan bilangan random untuk nilai setiap gen sesuai dengan representasi kromosom yang akan digunakan.
3. Pendekatan tertentu

   Cara ini dengan memasukkan nilai tertentu kedalam gen dari populasi awal yang dibentuk.
4. Permutasi Gen

   Cara ini dengan penggunaan permutasi josephus dalam kombinatorial.

Pada penelitian ini populasi awal yang akan dibentuk akan menggunakan metode *random generator*.

2.4.3.3 Nilai *Fitness*

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

Nilai *fitness* yang digunakan pada penelitian ini mengacu pada aturan dari permainan *puzzle* kakuro, dimana terdapat 2 aturan utama. Yang pertama adalah

aturan yang mengharuskan tidak ada nilai jawaban yang sama pada satu lajur. Untuk memenuhi aturan tersebut maka akan diterapkan nilai *fitness* dengan menggunakan persamaan 2.1 :

$$f_1 = \left(\frac{\mathrm{k} - \mathrm{e}}{k}\right) \qquad (2.1)$$

dimana,

f_1 = nilai *fitness* 1.

k = panjang kromosom.

e = jumlah gen yang menyalahi aturan permainan.

Panjang kromosom pada penelitian ini yaitu 40 untuk *puzzle* dengan ukuran 9 x 9 dan 104 pada ukuran 14 x 14. Nilai f_1 akan bernilai antara 0 sampai 1, dimana dengan nilai 1 tersebut merupakan kondisi nilai kromosom sudah memenuhi aturan tidak ada nilai jawaban yang sama pada satu lajur.

Untuk memenuhi aturan lainnya pada *puzzle* kakuro dimana jumlah nilai jawaban pada setiap lajur harus sama dengan nilai soalnya, maka akan diterapkan nilai *fitness* dengan menggunakan persamaan 2.2 :

$$f_2 = \frac{LB}{JL} \qquad (2.2)$$

dimana,

f_2 = nilai *fitness* 2.

LB = jumlah lajur yang benar dan memenuhi aturan.

JL = jumlah total lajur yang ada pada *puzzle* kakuro.

Jumlah lajur untuk permainan berukuran 9 x 9 yaitu 30 dan untuk ukuran 14 x 14 berjumlah 87 lajur. Proses penghitungan nilai f_2 ini hanya akan dilakukan jika nilai f_1 bernilai 1, jika nilai f_1 bernilai kurang dari 1 maka nilai f_2 akan diberi nilai 0. Nilai dari f_2 akan memiliki nilai antara 0 sampai 1, dimana nilai 1 tersebut merupakan kondisi nilai kromosom yang sudah memenuhi aturan dimana jumlah nilai jawaban pada setiap lajur harus sama dengan nilai soalnya.

Untuk menentukan apakah algoritma genetika berhasil menyelesaikan permainan *puzzle* kakuro, nilai dari *fitness* 1 dan *fitness* 2 akan dijumlahkan, jika nilai totalnya berjumlah 2 maka *puzzle* kakuro dinyatakan telah berhasil diselesaikan. Persamaan untuk mencari nilai *fitmess* total dapat dilihat pada persamaan 2.3.

$$f_{total} = f_1 + f_2 \qquad (2.3)$$

dimana,

f_{total} = nilai *fitness* total.

f_1 = nilai *fitness* 1.

f_2 = nilai *fitness* 2.

2.4.3.4 Seleksi

Setiap kromosom yang terdapat dalam populasi akan melalui proses seleksi untuk dipilih menjadi orang tua. Ada beberapa metode seleksi , diantaranya :

1. Seleksi *Roulette Wheel*

   Metode seleksi yang umum digunakan adalah seleksi *roulette wheel* , sesuai dengan namanya, metode ini menirukan permainan *roulette wheel*. Dimana masing-masing kromosom menempati potongan lingkaran pada roda *roulette wheel* secara proporsional sesuai dengan nilai *fitness*nya. Kromosom yang memiliki nilai *fitness* lebih besar menempati potongan lingkaran yang lebih besar dibandingkan kromosom bernilai *fitness* rendah.

2. Seleksi Ranking

   Seleksi ini memperbaiki proses seleksi yang sebelumnya yaitu *roulette wheel* karena pada seleksi tersebut kemungkinan selain satu kromosom mempunyai nilai fitness yang mendominasi hingga 90%. Nilai fitness yang lain akan mempunyai kemungkinan yang sangat kecil untuk terpilih. Seleksi rangking dipakai untuk mengatasi masalah dari kekurangan seleksi *roulette*

wheel. Pertama-tama, diurutkan seluruh kromosom berdasarkan bagus-tidaknya solusi berdasarkan nilai *fitness*-nya. Setelah diurutkan, kromosom terburuk diberi nilai fitness baru sebesar 1, kromosom kedua terburuk diberi nilai fitness baru sebesar 2, dan seterusnya. Kromosom terbaik diberi nilai fitness baru sebesar n dimana n adalah banyak kromosom dalam suatu populasi.

3. Seleksi *Steady State*

   Metode ini tidak banyak digunakan dalam proses seleksi karena dilakukan dengan mempertahankan individu yang terbaik. Pada setiap generasi, akan dipilih beberapa kromosom-kromosom yang memiliki nilai *fitness* terburuk akan digantikan dengan *offspring* yang baru. Sehingga pada generasi selanjutnya akan terdapat beberapa populasi yang dipertahankan.

4. Seleksi Turnamen

   Merupakan metode seleksi lainnya yang didasari fenomena alamiah seperti turnamen antar individu dalam populasi. Dilakukan dengan memilih secara acak beberapa kromosom dari populasi. Individu-individu yang terbaik dalam kelompok ini akan diseleksi sebagai induk.

5. *Truncation Random*

   Metode ini lebih mudah diterapkan jika dibandingkan dengan metode *roulette wheel*, pemilihan kromosom dilakukan secara acak tetapi tidak semua kromosom mendapatkan kesempatan tersebut, hanya kromosom terbaik saja yang berpeluang.

Pada penelitian ini metode seleksi yang akan digunakan adalah metode *ranking* yang merupakan perbaikan dari metode *roullete wheel.*

2.4.3.5 *Crossover* (Pindah Silang)

Sebuah kromosom yang mengarah pada solusi yang bagus dapat diperoleh dari proses memindah-silangkan dua buah kromosom. Pindah silang dapat berakibat buruk jika ukuran populasinya sangat kecil. Dalam suatu populasi yang sangat kecil, suatu kromosom dengan gen-gen yang mengarah ke solusi akan sangat cepat menyebar ke kromosom-kromosom lainnya. Untuk mengatasi

masalah tersebut digunakan sebuah aturan bahwa pindah silang hanya dapat dilakukan dengan sebuah probabilitas tertentu p_c. Pindah silang dapat dilakukan hanya jika suatu bilangan random [0,1] yang dibangkitkan kurang dari p_c yang ditentukan. Pada umumnya p_c diset mendekati 1 , misalnya 0,8.

Ada beberapa cara yang dapat digunakan untuk melakukan pindah silang sesuai dengan teknik pengkodean (*encoding)* yaitu sebagi berikut [**2**]:

1. *Binary encoding*
   a. Crossover satu titik

      Memilih satu titik tertentu, selanjutnya nilai biner sampai titik crossovernya dari induk pertama digunakan dan sisanya dilanjutkan dengan nilai biner dari induk kedua.

   b. *Crossover* N-titik

      Pada metode ini setiap kromosom induk dipotong menjadi N+1 bagian. Kromosom anak yang pertama dihasilkan dengan mewariskan bagian potongan urutan ganjil dari induk yang pertama dan mewariskan bagian potongan dalam urutan genap dari induk kedua. Demikian juga dengan kromosom anak yang kedua dihasilkan dengan cara yang sama dengan menggabungkan bagian yang tersisa dari kedua kromosom induknya.

   c. *Crossover uniform*

      Pada metode penyilangan ini, perlu dibangkitkan terlebih dahulu topeng penyilangan (*crossover mask*) yang terdiri atas kode biner sebanyak gen dalam kromosom. Gen-gen dalam kromosom anak ditentukan berdasarkan kode biner topeng penyilang. Untuk kromosom anak yang pertama, jika kode pada kedudukan topeng penyilangan bernilai satu, maka gen diwariskan dari induk pertama, sedangkan jika induk kode pada topeng penyilangan bernilai nol, maka gen diwariskan dari induk kedua.

2. *Permutation encoding*

   Memilih satu titik tertentu, nilai permutation sampai titik crossover. Pada induk pertama digunakan lalu sisanya dilakukan scan terlebih dahulu, jika

nilai permutasi pada induk kedua belum ada pada *offspring* nilai tersebut ditambahkan.

3. *Value encoding*

   Metode *crossover* pada pada pengkodean biner dapat digunakan.

4. *Tree encoding*

   Memilih satu titik tertentu dari tiap induk, dan menggunakan tree dibawah titik pada induk pertama dan tree dibawah induk kedua.

Pada penelitian ini metode crossover yang akan digunakan adalah *crossover* pada satu titik dengan nilai yang statis dan nilai dari probabilitas *crossover* yang digunakan adalah 0,6 yang merupakan rekomendasi dari De Jong [**9**].

2.4.3.6 Mutasi

Mutasi berperan untuk menggantikan informasi bit yang hilang akibat proses seleksi yang memungkinkan munculnya kembali gen yang tidak muncul pada insialisasi populasi. Berdasarkan bagian yang termutasi, proses mutasi dapat dibedakan atas dua bagian : [**3**]

1. Mutasi Pada Tingkat Kromosom

   Mutasi dengan metode ini akan mengubah seluruh nilai gen yang ada pada satu kromosom.

2. Mutasi Pada Tingkat Gen

   Mutasi dengan metode ini akan mengubah seluruh nilai yang ada pada suatu gen. Berdasarkan mutasi pada tingkat gen ini dapat dibagi menjadi 3 bagian yaitu :

   a. Swap Mutation

      Nilai dari 2 gen akan ditukar antara satu gen dan gen lainnya.

   b. 3 Swap Mutation

      Nilai dari 3 gen akan diputar, baik searah jarum jam ataupun berlawanan.

c. Insertion Mutation

Satu atau lebih nilai dimasukkan pada posisi tertentu dan nilai yang lain ikut berputar searah jarum jam ataupun berlawanan.

Pada penelitian ini metode mutasi yang akan digunakan adalah metode *swap mutation* dan nilai dari probabilitas mutasinya akan bernilai 0,001 yang merupakan rekomendasi dari De Jong [**9**].

2.4.3.7 Syarat Berhenti

Proses optimasi yang dilakukan oleh algoritma genetika akan berhenti setelah suatu syarat berhenti terpenuhi. Beberapa syarat berhenti yang biasa digunakan adalah batas nilai fungsi *fitness*, batas waktu komputasi, banyak generasi dan terjadinya konvergensi. Syarat berhenti yang sering digunakan adalah banyak generasi, tetapi tidak menutup kemungkinan untuk dipilih kombinasi beberapa syarat berhenti [**2**].

2.4.3.8 Penentuan Parameter Algoritma

Parameter algoritma merupakan salah satu bagian penting dalam penerapan algoritma gentika yang tidak mudah untuk ditentukan secara pasti. Tidak ada aturan yang pasti untuk menentukan parameter algoritma, baik probabilitas *crossover* (Pc), probabilitas mutasi (Pm), maupun ukuran populasi. Hal ini tidak terlepas dari prinsip algoritma genetika yang mengandalkan bilangan acak hampir dalam setiap langkahnya, mulai dari pembentukan populasi awal, proses penyilangan atau proses mutasi.

Pada penelitian ini probabilitas *crossover* yang akan digunakan bernilai 0,6, sedangkan untuk probabilitas mutasi yang digunakan bernilai 0,001 dan populasi yang akan digunakan berjumlah 10.

2.5 Pemograman Berorientasi Objek

Pemograman berorientasi objek adalah suatu strategi pembangunan perangkat lunak yang mengorganisasikan perangkat lunak sebagai kumpulan objek yang berisi data dan operasi yang diberlakukan terhadapnya. Menggunakan

pendekatan berorientasi objek akan memandang sistem yang akan dikembangkan sebagai suatu kumpuln objek yang berkorespondensi dengan objek dunia nyata.

Dalam rekayasa perangkat lunak, konsep pendekatan berorientasi objek dapat diterapkan pada tahap analisis, perancangan, pemograman, dan pengujian perangkat lunak. Berikut adaah beberapa konsep dasar yang harus dipahami pemograman berorientasi objek [**10**]:

1. Kelas (*Class*)

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

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

   Operasi atau metode pada sebuah kelas hampir sama dengan fungsi atau prosedur pada terstruktur. Sebuah kelas boleh memiliki lebih dari satu metode atau operasi. Metode atau operasi yang berfungsi untuk memanipulasi objek itu sendiri.
4. Atribut (*Attribute*)

   Atribut dari sebuah kelas adalah variabel global yang dimiliki sebuah kelas. Atribut dapat berupa nilai atau elemen-elemen data yang dimiliki oleh objek dalam kelas objek. Atribut secara individual oleh sebuah objek, misalnya berat, jenis, nama, dan sebagainya.

5. Abstraksi (*Abstraction*)

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

6. Enkapsulasi (*Encapsulation*)

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

7. Pewarisan (*Inheritance*)

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

8. Antarmuka (*Interface*)

   Antarmuka atau *interface* sangat mirip dengan kelas, tetapi tanpa atribut kelas dan tanpa memiliki metode yang dideklarasikan. Antarmuka biasanya digunakan agar kelas lain tidak langsung mengakses ke suatu kelas.

9. *Reusability*

   Pemanfaatan kembali objek yang sudah didefinisikan untuk suatu permasalahan pada permasalahan lainnya yang melibatkan objek tersebut.

10. Generalisasi dan Spesialisasi

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

11. Komunikasi Antar Objek

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

12. Polimorfisme (*Polymorphism*)

    Kemampuan suatu objek untuk digunakan dibanyak tujuan yang berbeda dengan nama yang sama sehingga menghemat baris program.

13. Package

Package adalah sebuah kontainer atau kemasan yang daoat digunakan untuk mengelompokkan kelas-kelas sehingga memungkinkan beberapa kelas yang bernama sama disimpan dalam package yang berbeda.

2.5.1 *Unified Modelling Language*

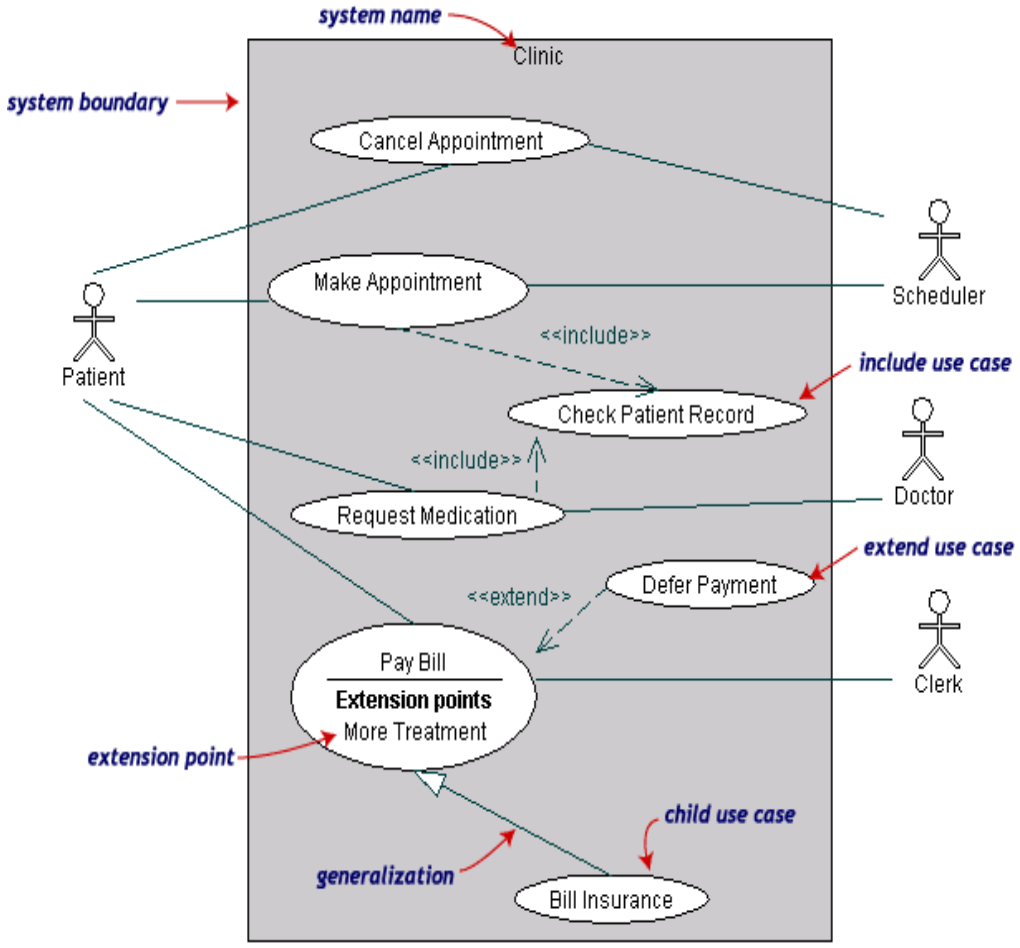
UML singkatan dari *Unified Modeling Language* yang berarti bahasa pemodelan standar. UML merupakan bahasa standar untuk merancang dan mendokumentasikan perangkat lunak dengan cara berorientasi objek. UML adalah bahasa grafis untuk mendokumentasi, menspesifikasi dan membangun sistem perangkat lunak. UML berorientasi objek, menerapkan banyak level abstraksi, tidak bergantung proses pengembangan, tidak bergantung bahasa pemrograman dan teknologi [**11**]. Ada beberapa diagram yang digunakan proses pembuatan perangkat lunak. Berikut diagram- diagram tersebut :

1. *Use case Diagram*

Diagram *use case* merupakan salah satu diagram untuk memodelkan aspek perilaku sistem. *Use case* mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan sistem yang akan dibuat [**11**]. *Use case* diagram digunakan untuk mengetahui fungsi apa saja yang terdapat pada sistem. Terdapat dua hal utama yang diperlukan dalam pembentukan suatu *use case* diagram yaitu aktor dan *use case.*

a. Aktor merupakan orang, benda maupun sistem lain yang berinteraksi dengan sistem yang akan dibangun.
b. *Use Case* merupakan fungsionalitas atau layanan yang disediakan oleh sistem.

Berikut adalah contoh dari penggunaan *use case diagram* pada gambar 2.6 :

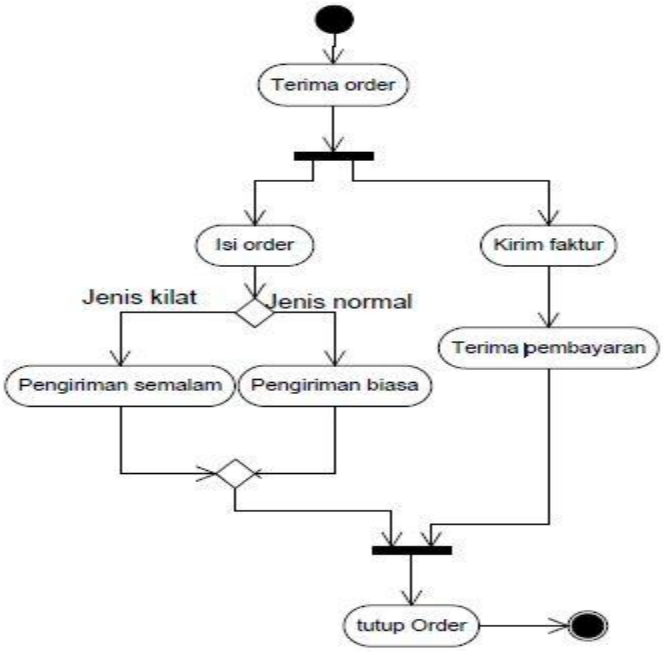


Gambar 2.6 Contoh Penggunaan *Use Case Diagram*

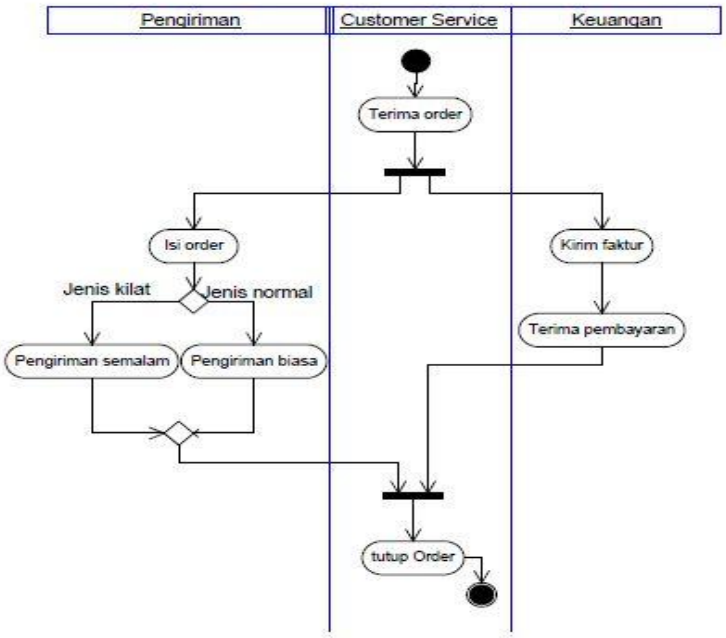
2. Activity Diagram

Activity Diagram menunjukkan *flow* aktifitas ke aktifitas (bukan status ke status). Activity diagram memodelkan *workflow* proses bisnis dan urutan aktifitas dalam sebuah prose. Diagram ini sangat mirip dengan flowchart. Membuat *activity diagram* pada awal pemodelan proses sangat membantu memahami keseluruhan proses. *Activity diagram* juga bermanfaat untuk menggambarkan paralel behavior atau menggambarkan interaksi antara beberapa use case.

Berikut adalah contoh dari penggunaan *activity diagram* pada gambar 2.7 dan 2.8 :



Gambar 2.7 Contoh Penggunaan *Activity Diagram* Tanpa *Swimlane*

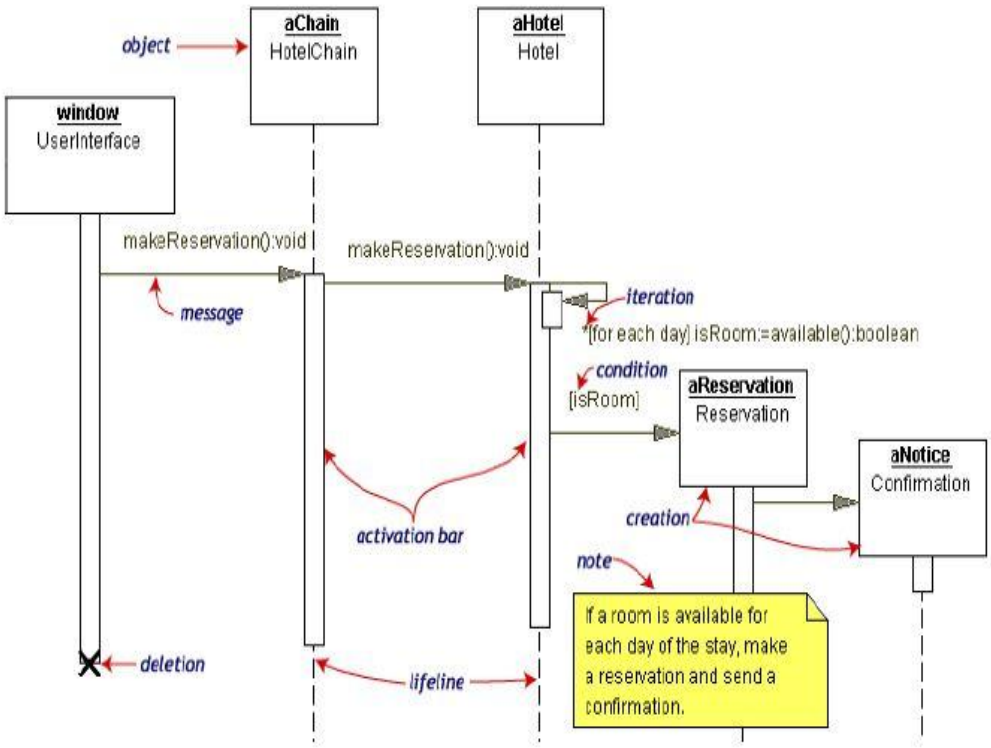


Gambar 2.8 Contoh Penggunaan *Activity Diagram* Dengan *Swimlane*

3. Sequence Diagram

Sequence diagram menggambarkan interaksi antara sejumlah objek dalam urutan waktu. Sequence diagram terdiri atas dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait). Umumnya sebuah sequence diagram menangkap behavior dari suatu skenario (*best case*) sebagai respons dari sebuah event untuk menghasilkan output tertentu. Diawali dari apa yang men-trigger aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan output apa yang dihasilkan.

Berikut adalah contoh dari penggunaan *sequence diagram* pada gambar 2.9 :



Gambar 2.9 Contoh Penggunaan *Sequence Diagram*

4. Class Diagram

Class diagram menunjukkan interaksi dan relasi antar class yang ada di dalam sistem. Sebuah class memiliki nama class, attribut, dan methods :

a. Nama class merupakan nama dari sebuah class yang digunakan dalam sebuah sistem.

b. Atribut merupakan variabel-variabel yang dimiliki oleh suatu class.
c. *Methods* merupakan fungsionalitas atau pekerjaan yang dilakukan oleh suatu class.

Atribut dan *methods* dapat memiliki salah satu sifat berikut :

a. *Private,* tidak dapat dipanggil dari luar *class* yang bersangkutan
b. *Protected,* hanya dapat dipanggil oleh *class* yang bersangkutan dan anak-anak yang mewarisinya.
c. *Public,* dapat dipanggil oleh siapa saja.

Class diagram menggambarkan relasi atau hubungan antar class dari sebuah sistem. Berikut ini beberapa gambaran relasi yang ada dalam class diagram :

a. *Asosiasi*

   Asosiasi merupakan hubungan antar *class* yang statis. *Class* yang mempunyai relasi asosiasi menggunakan *class* lain sebagai atribut pada dirinya.

b. *Agregasi*

   Agregasi merupakan relasi yang membuat class yang saling berelasi terikat satu sama lain namun tidak terlalu berkegantungan.

c. *Composition*

   Composition merupakan relasi agregasi dengan mengikat satu sama lain dengan ikatan yang sangat kuat dan saling berkegantungan.

d. *Dependency*

   Dependency merupakan hubungan antar class dimana class yang memiliki relasi dependency menggunakan class lain sebagai atribut pada method.

Kelas-kelas yang ada pada struktur sistem harus dapat melakukan fungsi-fungsi sesuai dengan kebutuhan sistem sehingga pembuat perangkat lunak atau *programmer* dapat membuat kelas-kelas di dalam program perangkat lunak sesuai dengan perancangan diagram kelas. Susunan struktur kelas yang baik pada diagram kelas sebaiknya memiliki jenis-jenis kelas berikut [**10**]:

a. Kelas *main*

   Kelas yang memiliki fungsi awal dieksekusi ketika sistem dijalankan.

b. Kelas yang menangani tampilan sistem (*view*)

   Kelas yang mendefinisikan dan mengatur tampilan pada *user*.

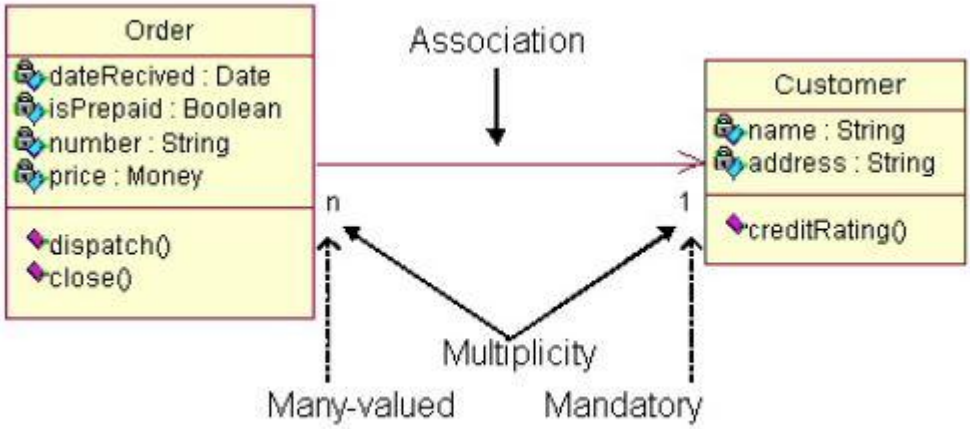
c. Kelas yang diambil dari pendefinisian *use case* (*controller*)

   Kelas yang menangani fungsi-fungsi yang harus ada diambil dari pendefinisian *use case*, kelas ini biasanya disebut dengan kelas proses yang menangani proses bisnis pada perangkat lunak.

d. Kelas yang diambil dari pendefinisian data (model)

   Kelas yang digunakan untuk memegang atau membungkus data menjadi sebuah sebuah kesatuan yang diambil maupun akan disimpan ke basis data. Semua tabel yang dibuat di basis data dapat dijadikan kelas, namun untuk tabel dari hasil relasi atau atribut *multivalue* pada ERD dapat dijadikan kelas tersendiri dapat juga tidak asalkan pengaksesannya dapat dipertanggungjawabkan atau tetap ada didalam perancangan kelas.

Berikut adalah contoh dari penggunaan *class diagram* pada gambar 2.6 :



Gambar 2.10 Contoh Penggunaan *Class Case Diagram*

2.6 C#

C# (dibaca: C sharp) merupakan sebuah bahasa pemrograman yang berorientasi objek yang dikembangkan oleh Microsoft sebagai bagian dari inisiatif kerangka .NET Framework. Bahasa pemrograman ini dibuat berbasiskan bahasa C++ yang telah dipengaruhi oleh aspek-aspek ataupun fitur bahasa yang terdapat pada bahasa-bahasa pemrograman lainnya seperti Java, Delphi, Visual Basic

dengan beberapa penyederhanaan. Menurut standar ECMA-334 C# *Language Specification*, nama C# terdiri atas sebuah huruf Latin C (U+0043) yang diikuti oleh tanda pagar yang menandakan angka # (U+0023). Tanda pagar # yang digunakan memang bukan tanda kres dalam seni musik (U+266F), dan tanda pagar # (U+0023) tersebut digunakan karena karakter kres dalam seni musik tidak terdapat di dalam keyboard standar [**12**].

Standar European Computer Manufacturer Association (ECMA) mendaftarkan beberapa tujuan desain dari bahasa pemrograman C#, sebagai berikut [**12**] :

1. Bahasa pemrograman C# dibuat sebagai bahasa pemrograman yang bersifat bahasa pemrograman *general-purpose*, berorientasi objek, modern, dan sederhana.
2. Bahasa pemrograman C# ditujukan untuk digunakan dalam mengembangkan komponen perangkat lunak yang mampu mengambil keuntungan dari lingkungan terdistribusi.
3. C# ditujukan agar cocok digunakan untuk menulis program aplikasi baik dalam sistem *client-server* maupun sistem *embedded*, mulai dari perangkat lunak yang sangat besar yang menggunakan sistem operasi yang canggih hingga kepada perangkat lunak yang sangat kecil yang memiliki fungsi-fungsi terdedikasi.

2.7 Perangkat Lunak Pendukung

Perangkat lunak pendukung merupakan perangkat lunak yang digunakan dalam mendukung pembangunan perangkat lunak. Berikut penjelasan dari perangkat lunak pendukung :

2.7.1 Microsoft Visual Studio

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

Kompiler yang dimasukkan ke dalam paket Visual Studio antara lain Visual C++, Visual C#, Visual Basic, Visual Basic .NET, Visual InterDev, Visual J++, Visual J#, Visual FoxPro, dan Visual SourceSafe [**13**].

Microsoft Visual Studio dapat digunakan untuk mengembangkan aplikasi dalam *native code* ataupun *managed code*. Selain itu, Visual Studio juga dapat digunakan untuk mengembangkan aplikasi Silverlight dan aplikasi Windows Mobile.