BAB II
LANDASAN TEORI

2.1 Tinjauan Pustaka

Tinjauan Pustaka yang telah dilakukan sejauh ini adalah dengan mempelajari beberapa hasil penelitian berupa jurnal dan makalah ilmiah. Khususnya yang terkait dengan topik-topik SAT *Problem*, SAT *Solver*, dan Algoritma Genetika. Berikut ini adalah rangkumannya:

1. SAT *Problem* dan SAT *Solver*

   Penelitian SAT *Problem* sebagian besar terfokus kepada permasalahan implementasinya dalam dunia nyata. Permasalahan dalam penelitian-penelitian SAT *Problem* biasanya adalah permasalahan NP-*Complete* (Zhao, Zhang, Ouyang, & Jiao, 2009). Apabila permasalahan dari sebuah penelitian adalah NP-*Complete* maka SAT *Problem* dapat merepresentasikan permasalahan tersebut dan penelitian tersebut pasti menggunakan SAT *Solver* untuk menyelesaikan permasalahan tersebut. Dalam berbagai penelitian SAT *Problem* yang dikerjakan oleh SAT *Solver* harus dalam bentuk CNF file (Zhao, Zhang, Ouyang, & Jiao, 2009).

   Penelitian mengenai SAT *Solver* sering kali terfokus kepada algoritma pembentuknya. Sudah banyak penelitian pengembangan SAT *Solver* dengan algoritma yang efektif dan efisien seperti, SAT *Solver* WalkSat & GRASP (Marques-Silva & Sakallah, 1996), Satz (Li & Anbulagan, 1997), zChaff (Vizel, Weissenbacher, & Malik, 2015) dan mChaff (Moskewicz, 2001). Algoritma yang dianggap terbaik dan konsisten untuk menyelesaikan SAT *Problem* adalah algoritma DPLL (Davis-Putnam-Logemann-Loveland). Hal itu dibuktikan dengan banyaknya SAT *Solver* dengan algoritma DPLL sering memenangkan SAT Competition, seperti SAT *Solver* yang menjadi juara kompetisi 2017 dibangun menggunakan penelitian pengembangan algoritma DPLL yaitu CDCL (*Conflict-Driven Clause Learning*) (Hamadi, Jabbour, & Saïs, 2016).

Hal itulah yang menyebabkan banyak penelitian menggunakan algoritma DPLL untuk menyelesaikan SAT *Problem* seperti yang penelitian yang dilakukan oleh Dicky Puja Pratama (Puja Pratama, 2018) dan Taufiq Hidayat & Agung Bahariyanto Irhasni (Hidayat & Bahariyanto Irhasni, SAT Solver dengan DPLL dalam Pemrograman, 2018). Namun belakangan ini penelitian mengenai algoritma pembentuk SAT *Solver* mengarah ke algoritma-algoritma metaheuristik seperti penelitian pengembangan SAT *Solver* menggunakan algoritma SVM (*Support vector machines*) (Huang, Li, & Li, 2019), ACO (*Ant Colony Optimization*) (Kho, Kasihmuddin, Mansor, & Sathasivam, 2019), dan BSO (*Bee Swarm Optimization*) (Sadeg, et al., June 2019). Bukan hanya algoritma saja yang mengalami perubahan arah dalam penelitian baru-baru ini bahasa pemrograman pun ikut mengalami perubahan. Dari yang penelitian-penelitian sebelumnya menggunakan bahasa pemrograman tingkat rendah sekarang penelitian-penelitian baru menggunakan bahasa pemrograman tingkat tinggi. Sebagai contoh penelitian untuk menyelesaikan SAT *Problem* dengan algoritma dengan bahasa pemrograman JAVA (Puja Pratama, 2018) dan penelitian untuk menyelesaikan SAT *Problem* dengan metode *sketching* dan menggunakan bahasa pemrograman JAVA (Hua, Zhang, Zhang, & Khurshid, 2019).

2. Algoritma Genetika

Penelitian Algoritma genetika terfokus pada dua hal. Pertama penelitian mengenai pengembangan algoritma genetika dan kedua penelitian mengenai implementasinya. Dalam penelitian pengembangan algoritma genetika sering kali algoritma genetika dimodifikasi dengan cara menggabungkannya dengan algoritma lain untuk menambah kecepatan optimasinya, seperti dalam percobaan penelitian penggabungan algoritma Taguchi dan Genetika (Vaghela & Prajapati, 2019). Sedangkan penelitian mengenai implementasi algoritma genetika seringkali digunakan untuk menyelesaikan permasalahan optimasi dalam dunia nyata seperti, penelitian untuk memecahkan permasalahan Optimalisasi konfigurasi lalu lintas udara (Sergeeva, Delahaye, Mancel, & Vidosavljevic, 2017) dan penelitian pengembangan manajemen aliran layanan dengan batasan anggaran dalam komputasi heterogen (AbdulHamed, Tawfeek, & Keshk, 2018).

2.2 Dasar Teori

Dasar teori dalam penyelesaian SAT *Problem* ini adalah teori yang digunakan sebagai acuan/pedoman dalam penelitian yang dilakukan. Seperti yang disebutkan dalam penjelasan di bawah ini:

2.2.1 SAT *Problem* (*Boolean Satisfiability Problem*) dan SAT *Solver*

SAT *Problem* adalah proses menentukan apakah sebuah formula adalah formula yang *satisfiable* atau *unsatisfiable*. Sebuah formula dapat dikatakan *satisfiabel* jika terdapat kombinasi nilai kebenaran komponen (variabel) pembentuknya yang membuat formula tersebut bernilai benar TRUE dan sebaliknya jika komponen penyusunnya membuat formula menjadi FALSE maka disebut *unsatisfiable* (Hidayat T. , 2013). Berikut contohnya:

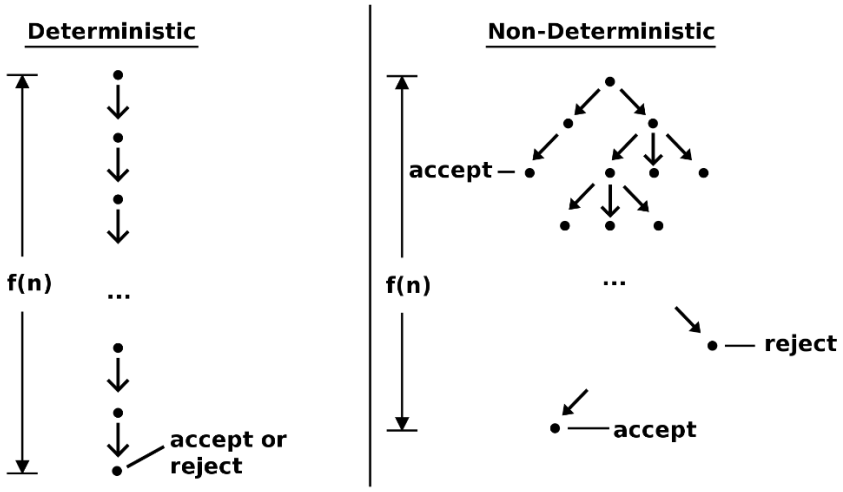
1. $a \wedge b$: *satisfiable*, karena jika a = T dan b = T maka formula menjadi TRUE.
2. $a \wedge (b \vee \neg c)$: *satisfiable*, karena jika a = T dan c = F maka formula menjadi TRUE.
3. $(a \vee b) \wedge b \wedge (\neg a \vee \neg c)$: *satisfiable*, karena jika a = F, b = T, dan c =F maka formula menjadi TRUE.
4. $a \wedge \neg a$: *unsatisfiable*, karena tiap kemungkinan variabel yang dimasukkan menjadikan formula menjadi FALSE.
5. $a \wedge b \wedge (\neg a \vee \neg b)$: *unsatisfiable*, karena tiap kemungkinan variabel yang dimasukkan menjadikan formula menjadi FALSE.

SAT *Problem* telah menjadi masalah penting dalam ilmu komputer sejak Stephen Cook membuktikan bahwa penyelesaian permasalahan ini tergolong dalam *Non Polynomial Complete* (NP-*Complete*) pada tahun 1971 (Vardi, 2014).

SAT *Solver* adalah *software* atau perangkat lunak yang digunakan untuk menyelesaikan SAT *Problem*, yaitu untuk menentukan apakah sebuah formula logika proposisi itu *satisfiable* atau *unsatisfiable*. Formula logika proposisi yang diselesaikan dengan SAT *Solver* harus dalam bentuk CNF (*Conjunctive Normal Form*) (Huth & Ryan, 2004).

2.2.2 NP-*Complete* (*Non deterministic Polynomial-Complete*)

Algoritma dapat digolongkan menjadi dua kelompok *deterministic* dan *nondeterministic.* Suatu algoritma dikatakan *deterministic* jika algortima tersebut dapat meneyelesaikan suatu permasalahan dengan langkah-langkah yang konstan. Algoritma *nondeterministic* adalah algoritma yang langkah-langkah peneyelesaian permasalahannya tidak konstan sekalipun dengan masukan yang sama. Ilustrasi algoritma *deterministic* dan *nondeterministic* dapat dilihat pada Gambar 2.1



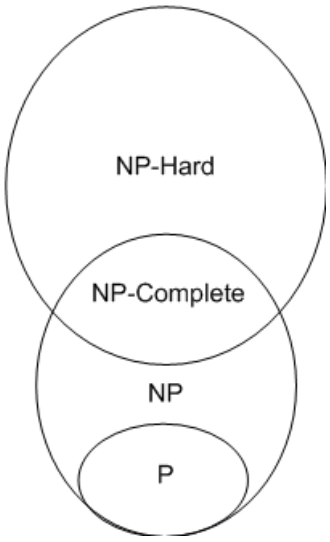
Gambar 2.1 Ilustrasi algoritma *deterministic* dan *nondeterministic*
Sumber: https://en.wikipedia.org/wiki/Nondeterministic_algorithm

Salah satu ukuran biaya dalam pengeksekusian sebuah algoritma adalah lamanya waktu (*time complexity*) yang diperlukan. Besarnya waktu yang diperlukan sebuah algoritma untuk menyelesaikan suatu permasalahan sebanding dengan jumlah masukan yang diberikan pada permasalahan tersebut (Weiss, 1996).

Waktu yang dibutuhkan suatu algoritma untuk menghasilkan solusi dari banyak permasalahan (*problem*) dapat dibagi menjadi tiga kelompok. Kelompok pertama disebut P (*deterministic Polynomial Problem*) merupakan permasalahan dimana tedapat suatu algoritma *deterministic* yang dapat menyelesaikan permasalahan tersebut dalam waktu polinomial. Contohnya adalah seperti permasalahan evaluasi polinomial dengan *O(n)* , pengurutan (*sorting*) dengan *O(n log n)* dan string editing dengan *O(mn)*. Kelompok kedua disebut NP (*Non deterministic Polynomial Problem*) yang merupakan permasalahan dimana tedapat suatu algoritma *nondeterministic* yang dapat menyelesaikan permasalahan tersebut dalam waktu polinomial (Darmawan & Nilamsari Kusumastuti, 2014). Kelompok terakhir disebut dengan NP-*Hard* (*Non deterministic Polynomial Problem Hard*). NP-*Hard* merupakan suatu

permasalahan dengan kompleksitas waktu penyelesaian permasalahannya adalah eksponensional.

Pada tahun 1971 Stephen Cook (Huth & Ryan, 2004) berhasil membuktikan penyelesaian permasalahan SAT *Problem* tergolong dalam NP-*Complete* (*Non deterministic Polynomial Problem Complete*). NP-*Complete* merupakan permasalahan yang dapat diselesaikan menggunakan algoritma *nondeterministic* dan merupakan NP-*Hard* (Karakashian & Puranda, 2008). Seluruh permasalahan NP-*Complete* merupakan permasalahan NP-*Hard* , tetapi sebagian permasalahan NP-*Hard* belum tentu menjadi permasalahan NP-*Complete* (Horowitz, Sahni, & Rajasekaran, 1998). Relasi antara P, NP, NP-*Complete* dan NP-*Hard* dapat dilihat pada Gambar 2.2.



Gambar 2.2 Relasi P, NP, NP-*Complete* dan NP-*Hard*

Sumber: https://www.researchgate.net/figure/Venn-diagram-for-the-various-complexity-groups_fig6_236679633

2.2.3 Formula proposisi dan CNF

Formula proposisi adalah formula dalam bentuk logika proposisi. Logika proposisi adalah logika yang disusun dari klausa/kalimat proposisi, yaitu klausa/kalimat yang dapat ditentukan nilai kebenarannya. Sebuah proposisi dinyatakan dalam sebuah variabel proposisi yang memiliki 2 kemungkinan nilai, benar (true) atau salah (false) (Hidayat T. , 2014).

Menurut buku berjudul "Logika Proposisi" (Hidayat T. , 2014) sebuah formula logika proposisi yang memenuhi syarat-syarat formula normal-form disebut dengan formula CNF (*Conjunctive Normal Form*). Syarat-syarat tersebut adalah sebagai berikut:

a. Hanya memiliki operator ∧, ∨, dan ¬.
b. Operator ¬ hanya boleh digunakan pada variabel proposisi.

CNF adalah salah satu bentuk formula yang termasuk formula dalam normal form karena memenuhi kedua syarat tersebut.

Secara formal, CNF didefinisikan seperti berikut ini. Sebuah formula X dikatakan dalam CNF jika formula X berbentuk:

$$X = K1 \wedge K2 \wedge K3 \wedge \ldots \wedge Kn$$

Dengan

1. $Ki = li1 \vee li2 \vee \ldots \vee li,mi$
2. $lij = p$ atau $\neg p$

p adalah proposisi, l disebut sebagai literal , dan K disebut sebagai klausa

Berikut ini adalah contoh bentuk formula-formula CNF (*Conjunctive Normal Form*). Untuk mempermudah pembacaan, jika literal pada sebuah klausa lebih dari satu maka literal tersebut ditulis dalam tanda kurung :

1. $a \wedge b$
2. $a \wedge (b \vee \neg c)$
3. $a \wedge b \wedge (a \vee \neg b)$
4. $(a \vee \neg b) \wedge (c \vee \neg d)$
5. $(a \vee \neg b) \wedge b \wedge (\neg a \vee \neg c)$
6. $(a \vee \neg b) \wedge (c \vee \neg d) \wedge (b \vee d)$
7. $(a \vee \neg b) \wedge (b \vee \neg c) \wedge (c \vee \neg a)$

2.2.4 Algoritma Genetika

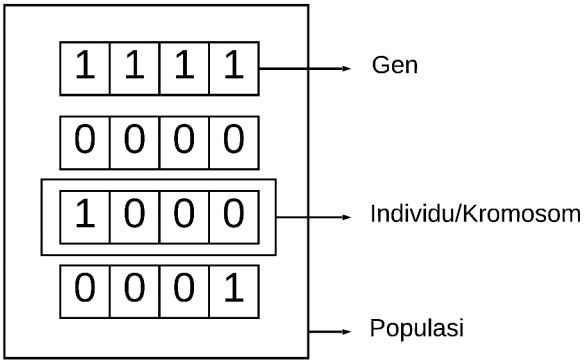
Algoritma Genetika adalah algoritma pencarian yang didasarkan pada mekanisme seleksi alamiah dan genetika alamiah (Suyanto, 2005). Sesuai awal konsepnya yaitu genetika, algoritma ini juga mengambil istilah-istilah yang ada dalam ilmu genetika seperti populasi, individu/kromosom, generasi, *selection*, *mutation*, dan *crossover*. Konsep yang ada dalam kaidah genetika ini diterapkan menjadi sebuah algoritma komputasi untuk menyelesaikan masalah yang membutuhkan constraint tinggi. Algoritma Genetika sebagai metode optimasi yang powerfull dimungkinkan telah menjadi teknik paling terkenal dalam bidang komputasi evolusioner pada saat ini (Gen & Cheng, 2000).

Secara umum, sebuah Algoritma Genetika memiliki lima komponen dasar, seperti yang ditulis dalam buku "Genetic Algorithm and Engineering Optimization" oleh Mitsuo Gen dan Runwei Cheng (Gen & Cheng, 2000).

- Representasi genetika untuk solusi masalah.
- Metode menciptakan inisiasi penyelesaian dari populasi.
- Evaluasi nilai *fitness* berdasarkan kemungkinan solusi.
- Metode genetika dalam penggantian keturunan dan reproduksi.
- Hasil akhir yang diharapkan dari pengolahan Algoritma Genetika.

a. Istilah dalam algoritma genetika

Berikut ini adalah ilustrasi singkat mengenai beberapa istilah yang ada pada algoritma genetika yaitu gen, individu/kromosom, dan populasi yang dapat dilihat pada Gambar 2.3.



Gambar 2.3 Penjelasan algoritma genetika

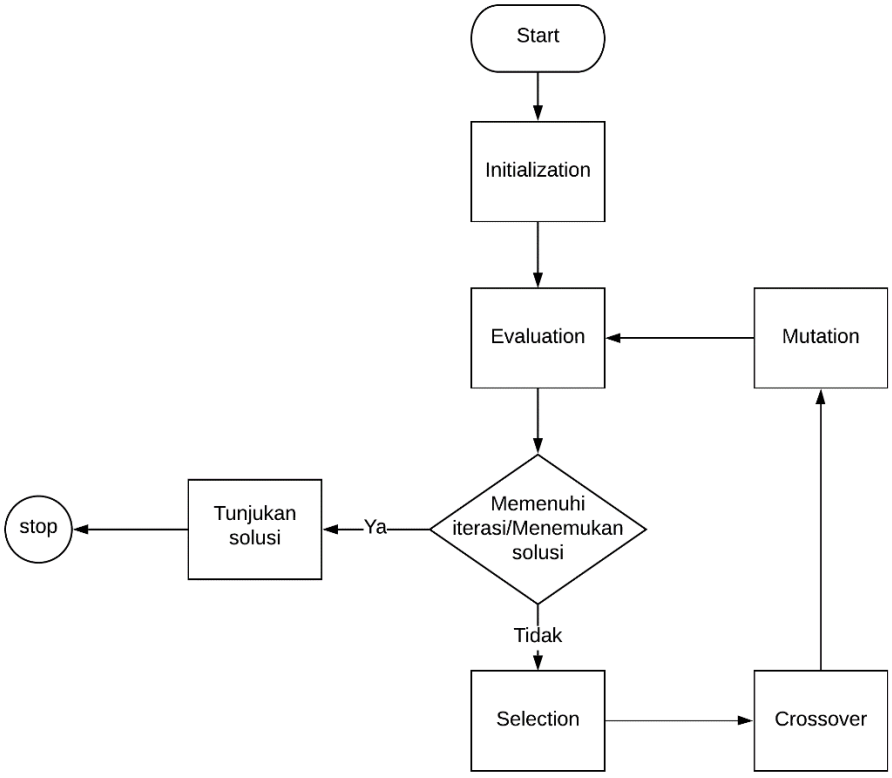
Karena Algoritma Genetika mengambil kaidah dari konsep genetika biologi, beberapa istilah dalam Algoritma Genetika juga mengambil konsep yang sama dengan genetika biologi. Berikut istilah-istilah dalam algoritma genetika yang diambil dari konsep genetika biologi dan penjelasanya:

- Gen/alel adalah susunan terkecil dari algoritma genetika. Gen tersusun dari karakter, simbol, bilangan ataupun karakter yang memiliki nilai tertentu dalam suatu permasalahan. Kumpulan gen menyusun sebuah kromosom.
- Kromosom atau Individu merupakan sebuah wujud representasi suatu solusi dari suatu permasalahan.

- Populasi adalah sekelompok individu yang akan dicari penyelesaiannya dalam Algoritma Genetika.
- Generasi adalah jumlah tingkatan peranakan sebuah kelompok populasi yang akan diseleksi, dikembangbiakkan, dan dimutasi beberapa kali sehingga menghasilkan generasi terbaik.
- Nilai *fitness* adalah nilai yang merepresentasikan kualitas suatu individu dalam suatu permasalahan.
- Fungsi kecocokan adalah fungsi/formula yang digunakan untuk menghitung nilai *fitness* suatu individu.

b. Mekanisme Algoritma Genetika

Secara umum, Algoritma Genetika dapat dijabarkan sebagai alur dan bagan pada yang dapat dilihat pada Gambar 2.4.



Gambar 2.4 Flowchart algoritma genetika

Sebelum Algoritma Genetika dijalankan, maka perlu didefinisikan fungsi *fitness* sebagai masalah yang ingin dioptimalkan. Jika nilai *fitness* semakin besar, maka sistem yang dihasilkan semakin baik. Fungsi *fitness* ditentukan dengan metode heuristik. Algoritma Genetika sangat tepat digunakan untuk penyelesaian masalah optimasi yang kompleks dan sukar diselesaikan dengan menggunakan metode konvensional. Sebagaimana halnya proses evolusi di alam, suatu Algoritma Genetika yang sederhana umumnya terdiri dari tiga operasi yaitu: operasi reproduksi, operasi crossover (persilangan), dan operasi mutasi. Struktur umum dari suatu Algoritma Genetika dapat didefinisikan dengan langkah-langkah sebagai berikut:

1. Membangkitkan populasi awal secara random.
2. Membentuk generasi baru dengan menggunakan tiga operasi di atas (seleksi, crossover, mutasi) secara berulang-ulang sehingga diperoleh kromosom yang cukup untuk membentuk generasi baru sebagai representasi dari solusi baru.
3. Evolusi solusi yang akan mengevaluasi setiap populasi dengan menghitung nilai *fitness* setiap kromosom hingga kriteria berhenti terpenuhi.

Bila kriteria berhenti belum terpenuhi maka akan dibentuk lagi generasi baru dengan mengulangi langkah regenerasi. Kriteria berhenti yang akan digunakan dalam penelitian ini adalah:

1. Berhenti pada generasi/iterasi yang telah ditentukan.
2. Berhenti setelah dalam beberapa generasi berturut-turut didapatkan nilai *fitness* paling optimum.

c. Metode *Tournament Strategy*

Metode *Tournament Strategy* adalah suatu metode yang menetapkan konsep elitisme pada sebuah populasi, dimana yang kuatlah yang akan bertahan. Artinya hanya individu-individu terbaik yang dapat melalui tahapan *selection*, *crossover*, dan *mutation*.

d. *Initialization*

Initialization atau inisialisasi adalah prosedur/tahapan yang digunakan untuk merancang struktur sebuah populasi dan membentuk populasi awal sesuai dengan rancangan yang tersebut.

e. *Evaluation*

Evaluation atau evaluasi adalah tahapan yang digunakan untuk menentukan kualitas individu/kromosom. Proses penentuan tersebut dilakukan dengan cara menghitung nilai *fitness* seluruh kromosom dalam populasi menggunakan fungsi kecocokan/objektif.

f. *Selection*

Selection/Seleksi adalah tahapan dimana akan dipilih individu-individu berdasarkan nilai *fitness* yang telah dihitung pada tahap *evaluation*. Individu-individu tersebut nantinya akan digunakan tahap *crossover*.

g. *Crossover*

Crossover atau perkawinan silang adalah prosedur memasangkan dua buah individu untuk kemudian dipisahkan masing-masing gennya dan dipasangkan dengan gen pasangannya. Sebuah individu dapat memperoleh solusi yang bagus jika dilakukan proses memindah-silangkan dua buah individu (Suyanto, 2005). *Crossover* menyediakan sebuah metode yang memungkinkan terjadinya eksplorasi bagian baru dalam ranah algoritma solusi (Coley, 2000)

h. *Mutation*

Dalam dunia nyata, sebuah *Mutation*/mutasi dapat terjadi akibat suatu proses. Begitu pula yang terjadi dalam Algoritma Genetika (Coley, 2000) Secara umum, proses mutasi dilakukan dengan cara membangkitkan sebuah bilangan random yang kurang dari probabilitas mutasi (mutation rate) kemudian gen yang ada diubah menjadi nilai kebalikannya. Semisal 0 menjadi 1, 1 menjadi 0. Prosedur ini telah disarankan oleh Suyanto dalam bukunya yang berjudul "Algoritma Genetika dalam Matlab" (Suyanto, 2005).

2.2.5 Solusi Pendekatan Iterasi

Solusi pendekatan, juga disebut Trial and Error, atau Trial and Improvement, digunakan untuk menghitung nilai ketika suatu persamaan/formula tidak dapat/sulit diselesaikan dengan menggunakan metode biasa. Prosesnya melibatkan memperkirakan nilai awal, memperoleh jawaban dari persamaan, dan kemudian meningkatkan perkirakan berikutnya. Proses ini diulangi hingga batas iterasi yang telah ditentukan tercapai.

2.2.6 JAVA

JAVA adalah bahasa pemrograman tingkat tinggi yang berorientasi objek. Program JAVA tersusun dari bagian yang disebut kelas. Kelas terdiri atas metode-metode yang melakukan pekerjaan dan mengembalikan informasi setelah melakukan tugasnya. Dalam setiap kelas juga terdapat atribut yang merupakan sifat berupa nilai atau kondisi yang dimiliki kelas tersebut.