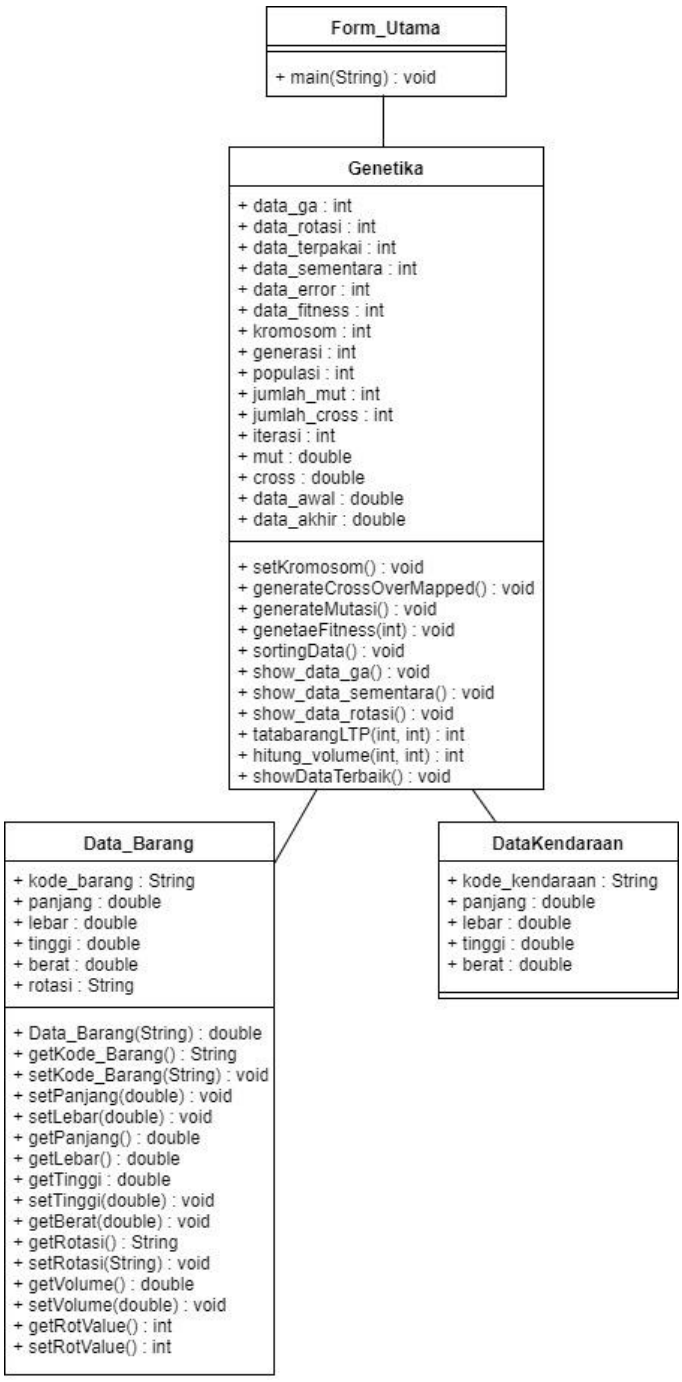
BAB 5 IMPLEMENTASI

5.1 Struktur Class

Struktur Class utama dapat dilihat pada Gambar 5.1.



Gambar 5.1 Struktur Class

Struktur Class pada Gambar 5.1 dapat dijelaskan pada Tabel 5.1.

Tabel 5.1 Keterangan Struktur Class

Class	Keterangan
Genetika	Class Genetika merupakan kelas utama dalam algoritme genetika. Yang berisi proses-proses algoritme genetika dari inisialisai kromosom awal, crossover, mutasi, perhitungan *fitness* dan proses tata letak barang.
Form_Utama	Class Form_Utama berisi untuk menampilkan komponen-komponen data barang, data kendaraan, parameter genetika dan hasil dari proses optimasi
Data_Barang	Class Data_Barang berisi dengan data barang berupa kode_barang, panjang, lebar, tinggi, berat, rotasi dan volume. Dalam Class ini juga berisi tentang pengaturan rotasi barang menjadi 6 variasi.
DataKendaraan	Class DataKendaraan berisi tentang data barang berupa panjang, lebar, tinggi, dan berat kendaraa.

5.2 Implementasi Algoritme

Implementasi algoritme genetika yang digunakan terdiri dari beberapa proses, seperti inisialisasi, crossover, mutasi, perhitungan *fitness* dan seleksi.

5.2.1 Proses Inisialisasi

Proses inisialisasi digunakan untuk membangkitkan populasi awal sejumlah populasi yang telah diinputkan. *Source code* proses inisialisasi dapat dilihat pada Gambar 5.1.

```
1   public void inisialisasiPopulasiAwal() {
2           Sistem.out.println("Kromosom: " + kromosom);
3           Sistem.out.println("Crossover Rate: " + cross);
4           Sistem.out.println("Mutation Rate: " + mut);
5
6           data_akhir = kromosom;
7           jumlah_cross = (int) (cross * populasi);
8           jumlah_mut = (int) (mut * populasi);
9           int total_data = populasi + jumlah_cross +
10  jumlah_mut;
11          data_ga = new int[total_data][kromosom];
12          data_rotasi = new int[total_data][kromosom];
13          data_terpakai = new int[total_data][kromosom];
14          data_error = new double[total_data];
15          data_fitness = new double[total_data];
16          data_sementara              =              new
17  int[total_data][kromosom];
18
19          Sistem.out.println("PopSize: " + populasi);
20          Sistem.out.println("Offspring Crossover Rate:
21  " + jumlah_cross);
22          Sistem.out.println("Offspring Mutation Rate: "
23  + jumlah_mut);
24          Sistem.out.println("");
25
26          for (int i = 0; i < populasi; i++) {
27              int[] krom = new int[kromosom];
28              for (int j = 0; j < kromosom; j++) {
29                  int data = 0;
30                  do {
31                      data            =            (int)
32  (rand.nextInt(kromosom) + data_awal);
33                  } while (cekInArray(krom, data));
34                  krom[j] = data;
35
36                  int rotasi = rand.nextInt(6) + 1;
37                  if
38  (barang[data].getRotasi().toLowerCase().equals("ya"))
39  {
40                      data_rotasi[i][j] = rotasi;
41                  } else {
42                      data_rotasi[i][j] = 1;
43                  }
44              }
45              data_ga[i] = krom;
46          }
47      }
48
```

Gambar 5.1 Proses Inisialisasi

Penjelasan *source code* pada Gambar 5.1 adalah sebagai berikut:

1. Baris ke-1 hingga ke-24 untuk proses inisialisasi.

2. Baris ke-27 hingga ke-35 merupakan proses pembangkitan kromosom awal secara random.
3. Baris ke-37 hingga ke-43 merupakan proses pembangkitan rotasi secara random sebanyak 6 variasi ketika barang dapat dirotasi. Dan bernilai variasi 1 ketika barang tidak dapat dirotasi.
4. Baris ke-47 menginputkan kromosom dan rotasi yang telah dibuat ke dalam individu.

5.2.2 Proses Crossover

Pada proses crossover digunakan untuk membentuk kromosom anak sejumlah *cr* dikali dengan jumlah populasi. Program ini menggunakan *partially mapped crossover. Source code* proses crossover terdapat dalam Gambar 5.2.

```
1     public void generateCrossOverMapped() {
2           int parent1[] = new int[jumlah_cross];
3           int parent2[] = new int[jumlah_cross];
4           int cross[] = new int[jumlah_cross];
5           int cross2[] = new int[jumlah_cross];
6
7           for (int i = 0; i < jumlah_cross ; i++) {
8               parent1[i]   =   (int)   (Math.random()   *
9   populasi); parent2[i] = 0;
10              do {
11                  parent2[i]  =  (int)  (Math.random()  *
12  populasi);
13              } while (parent1[i] == parent2[i]);
14          }
15
16          for (int i = 0; i < jumlah_cross ; i++) {
17              cross[i]   =   (int)   (Math.random()   *
18  kromosom); cross2[i] = 0;
19              do {
20                  cross2[i]  =  (int)  (Math.random()  *
21  kromosom);
22              } while (cross2[i] == cross[i]);
23          }
24
25          int[] hasil1 = new int[kromosom];
26          int[] hasil2 = new int[kromosom];
27          int[] rotasi1 = new int[kromosom], rotasi2 =
28  new int[kromosom];
29
30          for (int i = 0; i < jumlah_cross ; i++) {
31              for (int j = 0; j < kromosom; j++) {
32                  hasil1[j] = data_ga[parent1[i]][j];
33                  hasil2[j] = data_ga[parent2[i]][j];
34                  rotasi1[j]                              =
35  data_rotasi[parent1[i]][j];
36                  rotasi2[j]                              =
37  data_rotasi[parent2[i]][j];
38              }
39          }
40
41          for (int i = 0; i < jumlah_cross ; i++) {
42              for (int j = cross[i]; j < cross2[i]; j++)
43  {
44                  int temp = hasil1[j];
45                  hasil1[j] = hasil2[j];
46                  hasil2[j] = temp;
47
48                  int temprot = rotasi1[j];
49                  rotasi1[j] = rotasi2[j];
50                  rotasi2[j] = temprot;
51              }
52          }
53
```

```
54          for (int i = 0; i < jumlah_cross ; i++) {
55              int jumUtama = cross2[i] - cross[i];
56              Sistem.out.println("Parent1:"            +
57  (parent1[i] + 1) + "   parent2:" + (parent2[i] + 1) +
58  "   Cross:" + (cross[i] + 1));
59              showTest(hasil1, "\t");
60              showTest(hasil2, "\t");
61              for (int j = 0; j < kromosom; j++) {
62                  if (j >= cross[i] && j < cross2[i]) {
63                  } else {
64                      int pos1 = -1, pos2 = -1;
65                      for  (int  k  =  cross[i];  k  <
66  cross2[i]; k++) {
67                          if (j != k) {
68                              if      (hasil1[j]      ==
69  hasil1[k]) {
70                                  pos1 = k;
71                              }
72                              if      (hasil2[j]      ==
73  hasil2[k]) {
74                                  pos2 = k;
75                              }
76                          }
77                      }
78                      if (pos1 >= 0) {
79                          hasil1[j] = hasil2[pos1];
80                          rotasi1[j] = rotasi2[pos1];
81                      }
82                      if (pos2 >= 0) {
83                          hasil2[j] = hasil1[pos2];
84                          rotasi2[j] = rotasi1[pos2];
85                      }
86                  }
87              }
88              showTest(hasil1, "\t");
89              showTest(hasil2, "\t");
90              if (i * 2 < jumlah_cross) {
91                  data_ga[populasi + i * 2] = hasil1;
92                  data_rotasi[populasi  +  i  *  2]  =
93  rotasi1;
94              }
95
96              if ((i * 2 + 1) < jumlah_cross) {
97                  data_ga[populasi + i * 2 + 1] = hasil2;
98                  data_rotasi[populasi + i * 2 + 1] =
99  rotasi2;
100             }
101         }
102     }
103
```

Gambar 5.2 Proses Crossover

Penjelasan *source code* proses crossover adalah sebagai berikut:

1. Baris ke-2 hingga ke-5 sebagai inisialisasi.
2. Baris ke-7 hingga ke-14 untuk menentukan induk (parent 1 dan parent 2) secara random.
3. Baris ke-16 hingga ke-23 untuk menentukan substring pada masing-masing induk terpilih secara acak.
4. Baris ke-25 hingga ke-28 sebagai inisialisasi.
5. Baris ke-30 hingga ke-39 untuk menyimpan hasil induk dan rotasi.
6. Baris ke-41 hingga ke-52 untuk proses pertukaran pindah silang gen antara dua substring.
7. Baris ke-54 hingga ke-103 untuk proses pemetaan antara substring dan menentukan turunan anak.

5.2.3 Proses Mutasi

Pada proses mutasi digunakan untuk membentuk kromosom anak sejumlah *mr* dikali dengan jumlah populasi. Program ini menggunakan *random mutation. Source code* proses mutasi terdapat pada Gambar 5.3.

```
1   public void generateMutasi(){
2           for(int i=0; i<jumlah_mut; i++){
3               int poin1 = (int) (Math.random()*kromosom),
4   poin2=0;
5               do{
6                   poin2              =              (int)
7   (Math.random()*kromosom);
8               } while(poin1==poin2);
9               int          parent          =          (int)
10  (Math.random()*populasi);
11              Sistem.out.println("Parent:"+parent+"
12  point1:"+poin1+"   Poin2:"+poin2);
13              int[] hasil= new int[kromosom], rotasi =
14  new int[kromosom];
15
16              for (int j = 0; j < kromosom; j++) {
17                  hasil[j] =data_ga[parent][j];
18                  rotasi[j] = data_rotasi[parent][j];
19              }
20
21              int temp = hasil[poin1];
22              hasil[poin1] = hasil[poin2];
23              hasil[poin2] = temp;
24              data_ga[populasi+jumlah_cross+i] = hasil;
25
26              int trotasi = rotasi[poin1];
27              rotasi[poin1] = rotasi[poin2];
28              rotasi[poin2] = trotasi;
29              data_rotasi[populasi+jumlah_cross+i]     =
30  rotasi;
31          }
32      }
```

Gambar 5.3 Proses Mutasi

Penjelasan *source code* proses mutasi adalah sebagai berikut:

1. Baris ke-2 menentukan jumlah *offspring* mutasi.
2. Baris ke-3 hingga ke-7 menentukan dua titik potong gen dalam individu secara acak.
3. Baris ke-9 hingga ke-10 menentukan parent individu yang akan dimutasi.
4. Baris ke-16 hingga ke-18 untuk menyimpan parent individu dan rotasi terpilih.
5. Baris ke-21 hingga ke-24 merupakan proses menukar gen individu terpilih.
6. Baris ke-26 hingga ke-30 merupakan proses menukar gen rotasi terpilih.

5.2.4 Proses Perhitungan *Fitness*

Pada proses perhitungan nilai *fitness* digunakan untuk mengetahui individu mana yang optimal. *Source code* pr*oses* perhitungan *fitness* dapat dilihat dalam Gambar 5.4.

```
1  public void generateFitness(int iterasi){
2          double                    volumeBox                    =
3  DataKendaraan.panjang*DataKendaraan.lebar*DataKendaraan.tinggi;
4          int total_data= populasi+jumlah_cross+jumlah_mut;
5          for(int   i=((iterasi==0)?0:populasi);   i<total_data;
6  i++){
7              Sistem.out.println("Iterasi ke "+i);
8              int[]   terpakai   =   tatabarangLTP(data_ga[i],
9  data_rotasi[i]);
10             int volume = hitung_volume(data_ga[i],terpakai);
11             data_error[i] = volume;
12             data_terpakai[i] = terpakai;
13             data_fitness[i] = volume*100/volumeBox;
14         }
15     }
```

Gambar 5.4 Proses *Fitness*

Penjelasan *source code* menentukan *fitness* terbaik adalah sebagai berikut:

1. Baris ke-2 hingga ke-3 untuk menghitung volume kendaraan.
2. Baris ke-4 untuk menyimpan total data sebanyak jumlah populasi ditambah *offspring* crossover dan mutasi.
3. Baris ke-10 hingga ke-13 merupakan proses menghitung *fitness.*

5.2.5 Proses *Sorting* Data

Pada proses evaluasi terdapat proses perhitungan *sorting* data digunakan untuk mengurutkan data yang memiliki *fitness* terbesar ke terkecil. *Source code* proses *sorting* data dapat dilihat dalam Gambar 5.5.

```
1   public void sortingData() {
2           int total_data = populasi + jumlah_cross +
3   jumlah_mut;
4           for (int i = total_data; i > 0; i--) {
5               for (int j = 1; j < i; j++) {
6                   if    (data_fitness[j    -    1]    <
7   data_fitness[j]) {
8                       double temp = data_fitness[j - 1];
9                       data_fitness[j     -     1]     =
10  data_fitness[j];
11                      data_fitness[j] = temp;
12                      double temp1 = data_error[j - 1];
13                      data_error[j - 1] = data_error[j];
14                      data_error[j] = temp1;
15                      int[] temp_ga = data_ga[j - 1];
16                      data_ga[j - 1] = data_ga[j];
17                      data_ga[j] = temp_ga;
18                      int[] temp_rotasi = data_rotasi[j
19  - 1];
20                      data_rotasi[j     -     1]     =
21  data_rotasi[j];
22                      data_rotasi[j] = temp_rotasi;
23                      int[] temp_used = data_terpakai[j
24  - 1];
25                      data_terpakai[j     -     1]     =
26  data_terpakai[j];
27                      data_terpakai[j] = temp_used;
28                  }
29              }
30          }
31      }
```

Gambar 5.5 Proses *Sorting* Data

Penjelasan *source code* algoritme proses *sorting* data adalah sebagai berikut:

1. Baris ke-2 untuk menyimpan total data sebanyak jumlah populasi ditambah *offspring* crossover dan mutasi.
2. Baris ke-4 hingga ke-27 merupakan proses *sorting* data dari *fitness* terbesar ke terkecil sebanyak total data.

5.2.6 Proses Penyusunan Barang

Setelah mendapatkan individu terbaik dalam perhitungan algoritme genetika, kemudian dilakukan proses penyusunan barang. *Sorce code* proses penyusunan barang dapat dilihat dalam Gambar 5.6.

```
public int[] tatabarangLTP(int[] listbarang, int[] rotasi) {
        int[] terpakai = new int[listbarang.length];
        for (int i = 0; i < listbarang.length; i++) {
            terpakai[i] = 0;
        }
        int zpanjang = 0, zlebar = 0, ztinggi = 0, lbr = 0, tggi = 0, pjg = 0, zberat = 0, isMaxBerat = 0;
        do {
            ztinggi = tggi;
            pjg = 0;
            do {
                int count_brg = 0;
                zlebar = 0;
                tggi = 0;
                do {
                    lbr = 0;
                    if (terpakai[count_brg] == 0) {
                        Data_Barang data = barang[listbarang[count_brg]];

data.setRotValue(rotasi[count_brg]);
                        int tempPjg = (int) data.getPanjang();
                        int tempLbr = (int) data.getLebar();
                        int tempTg = (int) data.getTinggi();
                        int tempBerat = (int) data.getBerat();
                        lbr = tempLbr;
                        if (zberat + tempBerat <= DataKendaraan.berat) {
                            if (zlebar + tempLbr <= DataKendaraan.lebar) {
                                zlebar += tempLbr;
                                zberat += tempBerat;
                                if (pjg < tempPjg) {
                                    pjg = tempPjg;
                                }
                                if (tggi < tempTg) {
                                    tggi = tempTg;
                                }
                                terpakai[count_brg] = 1;
                            //Sistem.out.println("Data: "+zpanjang+":"+pjg+"  <>  "+ztinggi+":"+tggi+"  <>
```

```
54  "+zlebar+":"+lbr+"::    "+zberat+"::    "+tempBerat+"::
55  "+count_brg);
56                              }
57                          } else {
58                              isMaxBerat++;
59                            //Sistem.out.println("Berat
60  Melebihi Batas");
61                          }
62                      }
63                      count_brg++;
64                      if          (count_brg          ==
65  listbarang.length) {
66                          break; //cek_lbr==0 ||
67                      }
68                  } while (zlebar < DataKendaraan.lebar);
69                  if (tggi == 0) {
70                      break;
71                  }
72                  if      (ztinggi      +      tggi      <=
73  DataKendaraan.tinggi) {
74                      ztinggi += tggi;
75                      tggi = 0;
76                  } else {
77                      break;
78                  }
79              } while (ztinggi < DataKendaraan.tinggi);
80              if (pjg == 0) {
81                  break;
82              }
83              if       (zpanjang       +       pjg       <=
84  DataKendaraan.panjang) {
85                  zpanjang += pjg;
86              } else {
                    break;
                }
          } while (zpanjang < DataKendaraan.panjang);
          return terpakai;
      }
```

Gambar 5.6 Proses Penyusunan Barang

Penjelasan *source code* algoritme proses penyusunan barang adalah sebagai berikut:

1. Baris ke-19 hingga ke-31 digunakan untuk mengambil data parameter barang.
2. Baris ke-9 hingga ke-86 digunakan untuk menentukan kriteria barang yang dapat masuk ke dalam kendaraan, volume barang tidak melebihi volume kendaraan, Panjang barang tidak boleh melebih Panjang kendaraan, lebar barang tidak boleh melebihi lebar kendaraan, tinggi barang tidak boleh melebihi tinggi kendaraan, dan berat barang tidak boleh melebihi berat kendaraan.