PRAKTIKUM KECERDASAN BUATAN
TUGAS ALGORITMA GENETIKA

Kelompok 9

Shera Zahra Alya Nasywa Hardian (3122522026)

Muhammad Alief Putra Pratama (3122522023)

Subhan Fauzan (3122522029)

4 D3 PSDKU Sumenep

PRODI D3 TEKNIK INFORMATIKA

DEPARTEMEN TEKNIK INFORMATIKA DAN KOMPUTER

PENS PSDKU SUMENEP

PRAKTIKUM ALGORITMA GENETIKA

1. Buatlah kode program untuk WORD MATCHING

Algorithm.java

src > wordMatching > Algorithm.java > Algorithm > evolvePopulation(Population)

```java
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package wordMatching;

public class Algorithm {

    /* GA parameters */
    private static final double uniformRate = 0.5;
    private static final double mutationRate = 0.015;
    private static final int tournamentSize = 5;
    private static final boolean elitism = true;

    /* Public methods */
    // Evolve a population
    public static Population evolvePopulation(Population pop) {
        Population newPopulation = new Population(pop.size(), initialise:false);

        // Keep our best individual
        if (elitism) {
            newPopulation.saveIndividual(index:0, pop.getFittest());
        }

        // Crossover population
        int elitismOffset;
        if (elitism) {
            elitismOffset = 1;
        } else {
            elitismOffset = 0;
        }
        // Loop over the population size and create new individuals with
        // crossover
        for (int i = elitismOffset; i < pop.size(); i++) {
            Individual indiv1 = tournamentSelection(pop);
            Individual indiv2 = tournamentSelection(pop);
            Individual newIndiv = crossover(indiv1, indiv2);
            newPopulation.saveIndividual(i, newIndiv);
        }

        // Mutate population
        for (int i = elitismOffset; i < newPopulation.size(); i++) {
            mutate(newPopulation.getIndividual(i));
        }
```

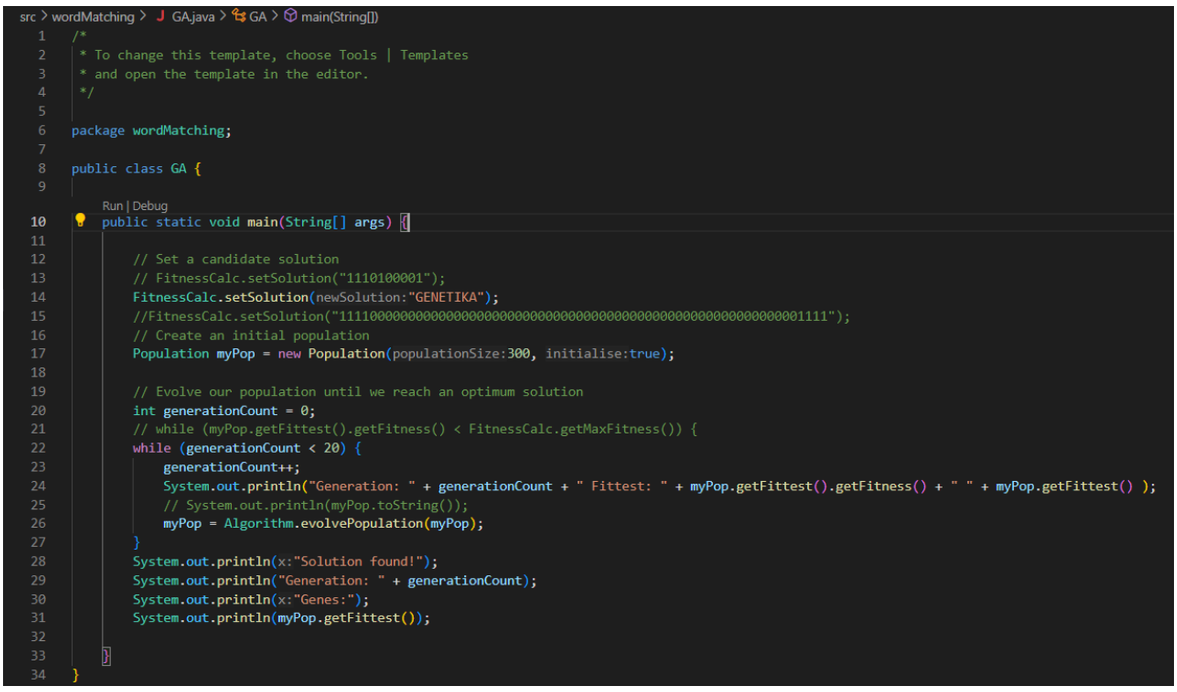
```java
        return newPopulation;
    }

    // Crossover individuals
    private static Individual crossover(Individual indiv1, Individual indiv2) {
        Individual newSol = new Individual();
        // Loop through genes
        for (int i = 0; i < indiv1.size(); i++) {
            // Crossover
            if (Math.random() <= uniformRate) {
                newSol.setGene(i, indiv1.getGene(i));
            } else {
                newSol.setGene(i, indiv2.getGene(i));
            }
        }
        return newSol;
    }

    // Mutate an individual
    private static void mutate(Individual indiv) {
        for (int i = 0; i < indiv.size(); i++) {
            if (Math.random() <= mutationRate) {
                // Create random gene
                byte gene = (byte) Math.round(Math.random());
                indiv.setGene(i, gene);
            }
        }
    }

    // Select individuals for crossover
    private static Individual tournamentSelection(Population pop) {
        Population tournament = new Population(tournamentSize, initialise:false);
        // For each place in the tournament get a random individual
        for (int i = 0; i < tournamentSize; i++) {
            int randomId = (int) (Math.random() * pop.size());
            tournament.saveIndividual(i, pop.getIndividual(randomId));
        }
        // Get the fittest
        Individual fittest = tournament.getFittest();
        return fittest;
    }
}
```

FitnessCalc.java

src > wordMatching > FitnessCalc.java > FitnessCalc > setSolution(String)

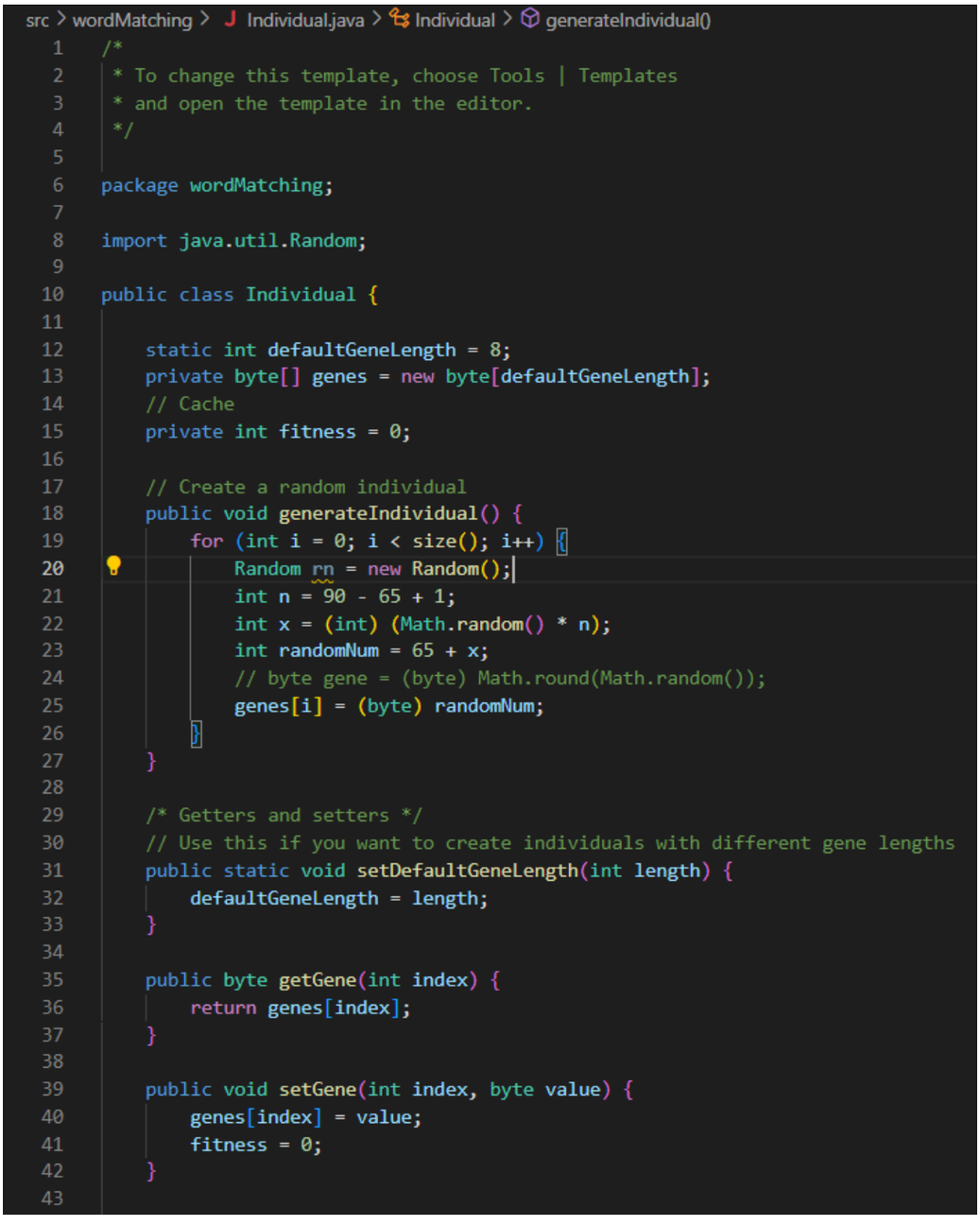
```java
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package wordMatching;

public class FitnessCalc {

    static byte[] solution = new byte[64];

    /* Public methods */
    // Set a candidate solution as a byte array
    public static void setSolution(byte[] newSolution) {
        solution = newSolution;
    }

    // To make it easier we can use this method to set our candidate solution
    // with string of 0s and 1s
    static void setSolution(String newSolution) {
        solution = new byte[newSolution.length()];
        solution = newSolution.getBytes();

    }

    // Calculate inidividuals fittness by comparing it to our candidate solution
    static int getFitness(Individual individual) {
        int fitness = 0;
        int selisih = 0;
        // Loop through our individuals genes and compare them to our cadidates
        for (int i = 0; i < individual.size() && i < solution.length; i++) {
            // if (individual.getGene(i) == solution[i]) {
            //     fitness++;
            // }
            selisih = selisih + Math.abs(individual.getGene(i) - solution[i]);
        }
        fitness = 26 * Individual.defaultGeneLength - selisih;
        return fitness;
    }

    // Get optimum fitness
    static int getMaxFitness() {
        int maxFitness = 26 * Individual.defaultGeneLength;
        return maxFitness;
    }
}
```

GA.java



```java
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package wordMatching;

public class GA {

    Run | Debug
    public static void main(String[] args) {

        // Set a candidate solution
        // FitnessCalc.setSolution("1110100001");
        FitnessCalc.setSolution(newSolution:"GENETIKA");
        //FitnessCalc.setSolution("1111000000000000000000000000000000000000000000000000000000001111");
        // Create an initial population
        Population myPop = new Population(populationSize:300, initialise:true);

        // Evolve our population until we reach an optimum solution
        int generationCount = 0;
        // while (myPop.getFittest().getFitness() < FitnessCalc.getMaxFitness()) {
        while (generationCount < 20) {
            generationCount++;
            System.out.println("Generation: " + generationCount + " Fittest: " + myPop.getFittest().getFitness() + " " + myPop.getFittest() );
            // System.out.println(myPop.toString());
            myPop = Algorithm.evolvePopulation(myPop);
        }
        System.out.println(x:"Solution found!");
        System.out.println("Generation: " + generationCount);
        System.out.println(x:"Genes:");
        System.out.println(myPop.getFittest());

    }
}
```

Individual.java



```java
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package wordMatching;

import java.util.Random;

public class Individual {

    static int defaultGeneLength = 8;
    private byte[] genes = new byte[defaultGeneLength];
    // Cache
    private int fitness = 0;

    // Create a random individual
    public void generateIndividual() {
        for (int i = 0; i < size(); i++) {
            Random rn = new Random();
            int n = 90 - 65 + 1;
            int x = (int) (Math.random() * n);
            int randomNum = 65 + x;
            // byte gene = (byte) Math.round(Math.random());
            genes[i] = (byte) randomNum;
        }
    }

    /* Getters and setters */
    // Use this if you want to create individuals with different gene lengths
    public static void setDefaultGeneLength(int length) {
        defaultGeneLength = length;
    }

    public byte getGene(int index) {
        return genes[index];
    }

    public void setGene(int index, byte value) {
        genes[index] = value;
        fitness = 0;
    }

```

```java
    /* Public methods */
    public int size() {
        return genes.length;
    }

    public int getFitness() {
        if (fitness == 0) {
            fitness = FitnessCalc.getFitness(this);
        }
        return fitness;
    }

    @Override
    public String toString() {
        String geneString = "";
        for (int i = 0; i < size(); i++) {
            geneString +=(char) getGene(i);
        }
        return geneString;
    }
}
```

Population.java

src > wordMatching > J Population.java > ...

```java
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package wordMatching;

public class Population {

    Individual[] individuals;

    /*
     * Constructors
     */
    // Create a population
    public Population(int populationSize, boolean initialise) {
        individuals = new Individual[populationSize];
        // Initialise population
        if (initialise==true) {
            // Loop and create individuals
            for (int i = 0; i < size(); i++) {
                Individual newIndividual = new Individual();
                newIndividual.generateIndividual();
                saveIndividual(i, newIndividual);
            }
        }
    }

    /* Getters */
    public Individual getIndividual(int index) {
        return individuals[index];
    }

    public Individual getFittest() {
        Individual fittest = individuals[0];
        // Loop through individuals to find fittest
        for (int i = 0; i < size(); i++) {
            if (fittest.getFitness() <= getIndividual(i).getFitness()) {
                fittest = getIndividual(i);
            }
        }
        return fittest;
    }

```

```java
    /* Public methods */
    // Get population size
    public int size() {
        return individuals.length;
    }

    // Save individual
    public void saveIndividual(int index, Individual indiv) {
        individuals[index] = indiv;
    }
    @Override
    public String toString() {
        String geneString = "";
        for (int i = 0; i < size(); i++) {
            geneString += getIndividual(i) + " " + getIndividual(i).getFitness()  + "\n";
        }
        return geneString;
    }
}
```

2. Pada laporan resmi, jelaskan bagian mana yang perlu diubah dari program dasar untuk mengimplementasikan WORD MATCHING

Perubahan yang perlu dilakukan pada masing-masing file adalah sebagai berikut:

FitnessCalc.java:

- Metode `setSolution` perlu diubah untuk menginisialisasi solusi dengan string alfabet. Panjang array solusi disesuaikan dengan panjang string solusi, dan setiap karakter dalam string solusi diubah menjadi byte yang mewakili karakter tersebut.

- Metode `getFitness` perlu diubah untuk menghitung fitness dengan membandingkan karakter dalam gen individu dengan karakter yang sesuai dalam solusi alfabet. Fitness dihitung berdasarkan jumlah karakter yang cocok antara individu dan solusi.

Individual.java:

- Metode `generateIndividual` perlu diubah untuk menghasilkan individu dengan karakter alfabet secara acak. Mengubah metode ini untuk menghasilkan byte yang mewakili karakter alfabet acak.

- Metode `toString` perlu diperbarui untuk mengonversi byte yang mewakili karakter alfabet menjadi string. Ini akan memungkinkan untuk mencetak individu dengan representasi karakter.

GA.java:

- Metode `main` perlu diubah untuk menginisialisasi dan menggunakan solusi sebagai string alfabet dalam `FitnessCalc.setSolution`. Panjang gen individu disesuaikan dengan panjang string solusi. Perubahan ini memungkinkan program untuk menghasilkan dan mengevaluasi individu dengan benar untuk pencocokan kata.

- Jumlah generasi mungkin perlu disesuaikan berdasarkan kompleksitas pencarian kata.

3. Berdasarkan kelompok 1-3, kata target : ALGORITMA

kelompok 4-6, kata target : MATCHING

kelompok 7-10, kata target : PRAKTIKUM

Karena kami kelompok 9 maka kata target yang digunakan adalah “PRAKTIKUM”

4. Lakukan percobaan dan isilah table dibawah ini

Percobaan	Jumlah Individu dalam populasi baru	Prob crossover	Prob mutasi	Jumlah iterasi	Output kata yang ditemukan	Output nilai fitness
1	50	0.5	0.05	10	PRAKTIKT	207 PRAKTIKT
2	100	0.5	0.05	10	PRAKTIJU	205 PRAKTIJU
3	150	0.5	0.05	10	PRAKTIKU	208 PRAKTIKU
4	200	0.9	0.1	20	PRAKTIKU	208 PRAKTIKU
5	250	0.9	0.1	20	PRAKTIMU	206 PRAKTIMU
6	300	0.9	0.1	20	PRAKTIKU	208 PRAKTIKU

BUKTI PERCOBAAN DENGAN KATA TARGET PRAKTIKUM

Percobaan 1

```
14      FitnessCalc.setSolution(newSolution:"PRAKTIKUM");
15      //FitnessCalc.setSolution("1111000000000000000000000000000000000000000000000000000001111");
16      // Create an initial population
17      Population myPop = new Population(populationSize:50, initialise:true);
18
19      // Evolve our population until we reach an optimum solution
20      int generationCount = 0;
21      // while (myPop.getFittest().getFitness() < FitnessCalc.getMaxFitness()) {
22      while (generationCount < 10) {
```

```
 9      /* GA parameters */
10      private static final double uniformRate = 0.5;
11      private static final double mutationRate = 0.05;
12      private static final int tournamentSize = 5;
13      private static final boolean elitism = true;
14
```

Hasil output:

```
PS C:\Users\shera\Downloads\GA (1)\GA>  c:; cd
ming\Code\User\workspaceStorage\bd3e910c60b649b
Generation: 1 Fittest: 180 PYAKYPDS
Generation: 2 Fittest: 187 OKAKSPNS
Generation: 3 Fittest: 194 PRAKPPJS
Generation: 4 Fittest: 198 QTAKTNKS
Generation: 5 Fittest: 199 QTAKYIKT
Generation: 6 Fittest: 204 QTAKTIKT
Generation: 7 Fittest: 206 PRAKTIKS
Generation: 8 Fittest: 207 PRAKTIKT
Generation: 9 Fittest: 207 PRAKTIKT
Generation: 10 Fittest: 207 PRAKTIKT
Solution found!
Generation: 10
Genes:
PRAKTIKT
```

Percobaan 2

```
14      FitnessCalc.setSolution(newSolution:"PRAKTIKUM");
15      //FitnessCalc.setSolution("1111000000000000000000000000000000000000000000000000000001111");
16      // Create an initial population
17      Population myPop = new Population(populationSize:100, initialise:true);
18
19      // Evolve our population until we reach an optimum solution
20      int generationCount = 0;
21      // while (myPop.getFittest().getFitness() < FitnessCalc.getMaxFitness()) {
22      while (generationCount < 10) {
```

```
 9      /* GA parameters */
10      private static final double uniformRate = 0.5;
11      private static final double mutationRate = 0.05;
12      private static final int tournamentSize = 5;
13      private static final boolean elitism = true;
14
```

Hasil output:

```
PS C:\Users\shera\Downloads\GA (1)\GA>  c:; cd
 'C:\Program Files\Java\jdk-20\bin\java.exe' '-
 '-cp' 'C:\Users\shera\AppData\Roaming\Code\Use
6073f4bcac65\redhat.java\jdt_ws\GA_3e0c407c\bin
Generation: 1 Fittest: 186 NNDMTMLO
Generation: 2 Fittest: 190 WTAKNHJT
Generation: 3 Fittest: 194 STCIUIHT
Generation: 4 Fittest: 194 TTCMTIHT
Generation: 5 Fittest: 200 NTAMTIJT
Generation: 6 Fittest: 200 NTAMTIJT
Generation: 7 Fittest: 201 ORBMTHLT
Generation: 8 Fittest: 203 NPAKTIJU
Generation: 9 Fittest: 205 PPAKTIJU
Generation: 10 Fittest: 205 PPAKTIJU
Solution found!
Generation: 10
Genes:
PRAKTIJU
```

Percobaan 3

```
14        FitnessCalc.setSolution(newSolution:"PRAKTIKUM");
15        //FitnessCalc.setSolution("1111000000000000000000000000000000000000000000000000000000001111");
16        // Create an initial population
17        Population myPop = new Population(populationSize:150, initialise:true);
18
19        // Evolve our population until we reach an optimum solution
20        int generationCount = 0;
21        // while (myPop.getFittest().getFitness() < FitnessCalc.getMaxFitness()) {
22        while (generationCount < 10) {
```

```
 9    /* GA parameters */
10    private static final double uniformRate = 0.5;
11    private static final double mutationRate = 0.05;
12    private static final int tournamentSize = 5;
13    private static final boolean elitism = true;
14
```

Hasil output:

```
PS C:\Users\shera\Downloads\GA (1)\GA>  c:; cd
 'C:\Program Files\Java\jdk-20\bin\java.exe' '-
 '-cp' 'C:\Users\shera\AppData\Roaming\Code\Use
6073f4bcac65\redhat.java\jdt_ws\GA_3e0c407c\bin
Generation: 1 Fittest: 181 PQCPNHGM
Generation: 2 Fittest: 192 PPCFRFKW
Generation: 3 Fittest: 196 PQCIRFKW
Generation: 4 Fittest: 198 PTAISFKW
Generation: 5 Fittest: 202 PRAKRFKT
Generation: 6 Fittest: 205 PQAKSIKT
Generation: 7 Fittest: 205 PRAKSHKT
Generation: 8 Fittest: 207 PRAKTHKU
Generation: 9 Fittest: 208 PRAKTIKU
Generation: 10 Fittest: 208 PRAKTIKU
Solution found!
Generation: 10
Genes:
PRAKTIKU
```

Percobaan 4

```
14        FitnessCalc.setSolution(newSolution:"PRAKTIKUM");
15        //FitnessCalc.setSolution("1111000000000000000000000000000000000000000000000000000000001111");
16        // Create an initial population
17        Population myPop = new Population(populationSize:200, initialise:true);
18
19        // Evolve our population until we reach an optimum solution
20        int generationCount = 0;
21        // while (myPop.getFittest().getFitness() < FitnessCalc.getMaxFitness()) {
22        while (generationCount < 20) {
```

```
    /* GA parameters */
    private static final double uniformRate = 0.9;
    private static final double mutationRate = 0.1;
    private static final int tournamentSize = 5;
    private static final boolean elitism = true;

```

Hasil output:

```
PS C:\Users\shera\Downloads\GA (1)\GA>  c:; cd '
  'C:\Program Files\Java\jdk-20\bin\java.exe' '-
  '-cp' 'C:\Users\shera\AppData\Roaming\Code\User
6073f4bcac65\redhat.java\jdt_ws\GA_3e0c407c\bin'
Generation: 1 Fittest: 180 QVHLOJHO
Generation: 2 Fittest: 180 QVHLOJHO
Generation: 3 Fittest: 183 PMBOMHOX
Generation: 4 Fittest: 183 PMBOMHOX
Generation: 5 Fittest: 193 ORBOTGKN
Generation: 6 Fittest: 193 ORBOTGKN
Generation: 7 Fittest: 196 OPALWHOU
Generation: 8 Fittest: 196 ORBOUGKX
Generation: 9 Fittest: 202 PRBKTGKX
Generation: 10 Fittest: 202 PRBKTGKX
Generation: 11 Fittest: 202 PRBKTGKX
Generation: 12 Fittest: 204 PRAKTHKX
Generation: 13 Fittest: 206 PRBKTHKU
Generation: 14 Fittest: 206 PRBKTHKU
Generation: 15 Fittest: 206 PRBKTHKU
Generation: 16 Fittest: 208 PRAKTIKU
Generation: 17 Fittest: 208 PRAKTIKU
Generation: 18 Fittest: 208 PRAKTIKU
Generation: 19 Fittest: 208 PRAKTIKU
Generation: 20 Fittest: 208 PRAKTIKU
Solution found!
Generation: 20
Genes:
PRAKTIKU
```

Percobaan 5

```
    FitnessCalc.setSolution(newSolution:"PRAKTIKUM");
    //FitnessCalc.setSolution("1111000000000000000000000000000000000000000000000000000000000001111");
    // Create an initial population
    Population myPop = new Population(populationSize:250, initialise:true);

    // Evolve our population until we reach an optimum solution
    int generationCount = 0;
    // while (myPop.getFittest().getFitness() < FitnessCalc.getMaxFitness()) {
    while (generationCount < 20) {
```

```
    /* GA parameters */
    private static final double uniformRate = 0.9;
    private static final double mutationRate = 0.1;
    private static final int tournamentSize = 5;
    private static final boolean elitism = true;

```

Hasil output:

```
PS C:\Users\shera\Downloads\GA (1)\GA>  c:; cd
  'C:\Program Files\Java\jdk-20\bin\java.exe' '
  '-cp' 'C:\Users\shera\AppData\Roaming\Code\Us
6073f4bcac65\redhat.java\jdt_ws\GA_3e0c407c\bi
Generation: 1 Fittest: 181 PRDLNBPZ
Generation: 2 Fittest: 188 TNFKTKPU
Generation: 3 Fittest: 192 WRAITMIT
Generation: 4 Fittest: 192 WRAITMIT
Generation: 5 Fittest: 195 NRAKQKHR
Generation: 6 Fittest: 197 NRAKQIHR
Generation: 7 Fittest: 198 NRAKTMHT
Generation: 8 Fittest: 199 QRAKVIMY
Generation: 9 Fittest: 200 PRAKTKHR
Generation: 10 Fittest: 203 NRAKTIIT
Generation: 11 Fittest: 203 ORAKTJIT
Generation: 12 Fittest: 204 ORAKTIIT
Generation: 13 Fittest: 204 ORAKTIIT
Generation: 14 Fittest: 204 ORAKTIMT
Generation: 15 Fittest: 205 PRAKTIHU
Generation: 16 Fittest: 205 PRAKTIHU
Generation: 17 Fittest: 205 PRAKTIHU
Generation: 18 Fittest: 205 ORAKTIIU
Generation: 19 Fittest: 205 ORAKTIIU
Generation: 20 Fittest: 206 PRAKTIIU
Solution found!
Generation: 20
Genes:
PRAKTIMU
```

Percobaan 6

```
14          FitnessCalc.setSolution(newSolution:"PRAKTIKUM");
15          //FitnessCalc.setSolution("1111000000000000000000000000000000000000000000000000000000001111");
16          // Create an initial population
17          Population myPop = new Population(populationSize:300, initialise:true);
18
19          // Evolve our population until we reach an optimum solution
20          int generationCount = 0;
21          // while (myPop.getFittest().getFitness() < FitnessCalc.getMaxFitness()) {
22          while (generationCount < 20) {
```

```
9       /* GA parameters */
10      private static final double uniformRate = 0.9;
11      private static final double mutationRate = 0.1;
12      private static final int tournamentSize = 5;
13      private static final boolean elitism = true;
14
```

Hasil output:

```
PS C:\Users\shera\Downloads\GA (1)\GA>  c:; cd
ming\Code\User\workspaceStorage\bd3e910c60b649b
Generation: 1 Fittest: 185 ZVAMQHKX
Generation: 2 Fittest: 186 HPHIRIJU
Generation: 3 Fittest: 192 HPBIRIJU
Generation: 4 Fittest: 193 HQBIRIJU
Generation: 5 Fittest: 193 HQBIRIJU
Generation: 6 Fittest: 193 HQBIRIJU
Generation: 7 Fittest: 200 NQBIRIKU
Generation: 8 Fittest: 200 NQBIRIKU
Generation: 9 Fittest: 200 NQBIRIKU
Generation: 10 Fittest: 201 ORAKTHKP
Generation: 11 Fittest: 202 NQBITIKU
Generation: 12 Fittest: 203 OQBITIKU
Generation: 13 Fittest: 203 OPBKTHKU
Generation: 14 Fittest: 206 ORAKTHKU
Generation: 15 Fittest: 206 ORAKTHKU
Generation: 16 Fittest: 206 ORAKTHKU
Generation: 17 Fittest: 207 ORAKTIKU
Generation: 18 Fittest: 207 PRAKTHKU
Generation: 19 Fittest: 207 ORAKTIKU
Generation: 20 Fittest: 208 PRAKTIKU
Solution found!
Generation: 20
Genes:
PRAKTIKU
```