package ga1;

import java.util.Random;

public class GA1 {

Population population = new Population();

Individual fittest;

Individual secondFittest;

int generationCount = 0;

public static void main(String[] args) {

Random rn = new Random();

GA1 demo = new GA1();

//Initialize population

demo.population.initializePopulation(10);

//Calculate fitness of each individual

demo.population.calculateFitness();

System.out.println("Generation: " + demo.generationCount + " Fittest: " + demo.population.fittest);

//While population gets an individual with maximum fitness

while (demo.population.fittest < 5) {

++demo.generationCount;

//Do selection

demo.selection();

//Do crossover

demo.crossover();

//Do mutation under a random probability

if (rn.nextInt()%7 < 5) {

demo.mutation();

}

//Add fittest offspring to population

demo.addFittestOffspring();

//Calculate new fitness value

demo.population.calculateFitness();

System.out.println("Generation: " + demo.generationCount + " Fittest: " + demo.population.fittest);

}

System.out.println("\nSolution found in generation " + demo.generationCount);

System.out.println("Fitness: "+demo.population.getFittest().fitness);

System.out.print("Genes: ");

for (int i = 0; i < 5; i++) {

System.out.print(demo.population.getFittest().genes[i]);

}

System.out.println("");

}

//Selection

void selection() {

//Select the most fittest individual

fittest = population.getFittest();

//Select the second most fittest individual

secondFittest = population.getSecondFittest();

}

//Crossover

void crossover() {

Random rn = new Random();

//Select a random crossover point

int crossOverPoint = rn.nextInt(population.individuals[0].geneLength);

//Swap values among parents

for (int i = 0; i < crossOverPoint; i++) {

int temp = fittest.genes[i];

fittest.genes[i] = secondFittest.genes[i];

secondFittest.genes[i] = temp;

}

}

//Mutation

void mutation() {

Random rn = new Random();

//Select a random mutation point

int mutationPoint = rn.nextInt(population.individuals[0].geneLength);

//Flip values at the mutation point

if (fittest.genes[mutationPoint] == 0) {

fittest.genes[mutationPoint] = 1;

} else {

fittest.genes[mutationPoint] = 0;

}

mutationPoint = rn.nextInt(population.individuals[0].geneLength);

if (secondFittest.genes[mutationPoint] == 0) {

secondFittest.genes[mutationPoint] = 1;

} else {

secondFittest.genes[mutationPoint] = 0;

}

}

//Get fittest offspring

Individual getFittestOffspring() {

if (fittest.fitness > secondFittest.fitness) {

return fittest;

}

return secondFittest;

}

//Replace least fittest individual from most fittest offspring

void addFittestOffspring() {

//Update fitness values of offspring

fittest.calcFitness();

secondFittest.calcFitness();

//Get index of least fit individual

int leastFittestIndex = population.getLeastFittestIndex();

//Replace least fittest individual from most fittest offspring

population.individuals[leastFittestIndex] = getFittestOffspring();

}

}

//Individual class

class Individual {

int fitness = 0;

int[] genes = new int[5];

int geneLength = 5;

public Individual() {

Random rn = new Random();

//Set genes randomly for each individual

for (int i = 0; i < genes.length; i++) {

genes[i] = Math.abs(rn.nextInt() % 2);

}

fitness = 0;

}

//Calculate fitness

public void calcFitness() {

fitness = 0;

for (int i = 0; i < 5; i++) {

if (genes[i] == 1) {

++fitness;

}

}

}

}

//Population class

class Population {

int popSize = 10;

Individual[] individuals = new Individual[10];

int fittest = 0;

//Initialize population

public void initializePopulation(int size) {

for (int i = 0; i < individuals.length; i++) {

individuals[i] = new Individual();

}

}

//Get the fittest individual

public Individual getFittest() {

int maxFit = Integer.MIN\_VALUE;

int maxFitIndex = 0;

for (int i = 0; i < individuals.length; i++) {

if (maxFit <= individuals[i].fitness) {

maxFit = individuals[i].fitness;

maxFitIndex = i;

}

}

fittest = individuals[maxFitIndex].fitness;

return individuals[maxFitIndex];

}

//Get the second most fittest individual

public Individual getSecondFittest() {

int maxFit1 = 0;

int maxFit2 = 0;

for (int i = 0; i < individuals.length; i++) {

if (individuals[i].fitness > individuals[maxFit1].fitness) {

maxFit2 = maxFit1;

maxFit1 = i;

} else if (individuals[i].fitness > individuals[maxFit2].fitness) {

maxFit2 = i;

}

}

return individuals[maxFit2];

}

//Get index of least fittest individual

public int getLeastFittestIndex() {

int minFitVal = Integer.MAX\_VALUE;

int minFitIndex = 0;

for (int i = 0; i < individuals.length; i++) {

if (minFitVal >= individuals[i].fitness) {

minFitVal = individuals[i].fitness;

minFitIndex = i;

}

}

return minFitIndex;

}

//Calculate fitness of each individual

public void calculateFitness() {

for (int i = 0; i < individuals.length; i++) {

individuals[i].calcFitness();

}

getFittest();

}

}

**Run this program and get the output. Explain:**

**1. How the algorithm produces the population such as gene and chromosome.**

* The algorithm declares the Population. Inside the Population got 10 individuals, the population will increase until the fittest equal to 5.

**2. How the algorithm calculates the fitness function.**

* In the calculateFitness function, it will calculate the fitness of individual in population. The individual fitness can be count using calcFitness function. In this function, if gene equal to 1, fitness will increase by one. In the getFittest function, it will get the maximum fitness of the population.