LAB-9

QUESTION:

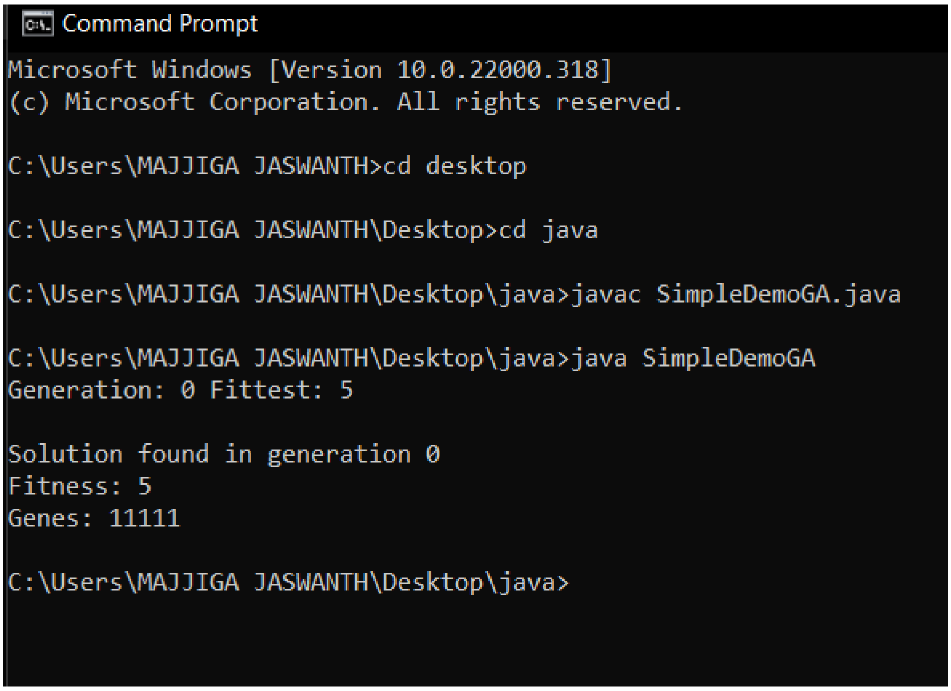
Write a program to implement genetic algorithm optimization technique

CODE:

```
import java.util.Random;
public class SimpleDemoGA {


    Population population = new Population();
    Individual fittest;
    Individual secondFittest;
    int generationCount = 0;
    public static void main(String[] args) {
        Random rn = new Random();
        SimpleDemoGA demo = new SimpleDemoGA();
        demo.population.initializePopulation(10);
        demo.population.calculateFitness();
        System.out.println("Generation: " +
demo.generationCount + " Fittest: " +
demo.population.fittest);
        while (demo.population.fittest < 5) {
            ++demo.generationCount;
            demo.selection();
            demo.crossover();
            if (rn.nextInt()%7 < 5) {
                demo.mutation();
            }
            demo.addFittestOffspring();
            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("");
    }
    void selection() {
        fittest = population.getFittest();
        secondFittest = population.getSecondFittest();
    }
    void crossover() {
        Random rn = new Random();
        int crossOverPoint =
rn.nextInt(population.individuals[0].geneLength);
        for (int i = 0; i < crossOverPoint; i++) {
            int temp = fittest.genes[i];
            fittest.genes[i] = secondFittest.genes[i];
            secondFittest.genes[i] = temp;
        }
    }
    void mutation() {
        Random rn = new Random();
        int mutationPoint =
rn.nextInt(population.individuals[0].geneLength);
        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;
        }
    }
    Individual getFittestOffspring() {
        if (fittest.fitness > secondFittest.fitness) {
            return fittest;
        }
        return secondFittest;
    }
    void addFittestOffspring() {
        fittest.calcFitness();
        secondFittest.calcFitness();
        int leastFittestIndex = 
population.getLeastFittestIndex();
        population.individuals[leastFittestIndex] = 
getFittestOffspring();
    }
}
class Individual {
    int fitness = 0;
    int[] genes = new int[5];
    int geneLength = 5;
    public Individual() {
        Random rn = new Random();
        for (int i = 0; i < genes.length; i++) {
            genes[i] = Math.abs(rn.nextInt() % 2);
        }
        fitness = 0;
    }
    public void calcFitness() {
        fitness = 0;
        for (int i = 0; i < 5; i++) {
            if (genes[i] == 1) {
                ++fitness;
            }
        }
    }
```

```
}
class Population {
    int popSize = 10;
    Individual[] individuals = new Individual[10];
    int fittest = 0;
    public void initializePopulation(int size) {
        for (int i = 0; i < individuals.length; i++) {
            individuals[i] = new 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];
    }
    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];
    }
    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;
    }
    public void calculateFitness() {

        for (int i = 0; i < individuals.length; i++) {
            individuals[i].calcFitness();
        }
        getFittest();
    }


}
```

OUTPUT :



```
Microsoft Windows [Version 10.0.22000.318]
(c) Microsoft Corporation. All rights reserved.

C:\Users\MAJJIGA JASWANTH>cd desktop

C:\Users\MAJJIGA JASWANTH\Desktop>cd java

C:\Users\MAJJIGA JASWANTH\Desktop\java>javac SimpleDemoGA.java

C:\Users\MAJJIGA JASWANTH\Desktop\java>java SimpleDemoGA
Generation: 0 Fittest: 5

Solution found in generation 0
Fitness: 5
Genes: 11111

C:\Users\MAJJIGA JASWANTH\Desktop\java>
```