LAB : 3

OBJECTIVE :

WAP to implement the genetic algorithm.

Requirements :

- Windows/ Mac / Linux Pc
- JDK installed (here using JDK 15)
- Text Editor / IDE (here using VS Code)

Problem Statement :

Create a random pouulation of size ‘n’ , where each individual (technically a chromosome, which is a solution also) is represented by a binary string of 0’s and 1’s (initialized randomly). In other words each gene can have two values 0 and 1 , and string of ‘n’ genes represents an individual. We nedd to apply Algorithm (GA) to get a solution with all 1’s.

Implementation :

Individual.java :

```
package Lab3;
import java.lang.Math;

public class Individual {
    private int[] chromosome;
    private double fitness = -1;

    public Individual(int[] chromosome) {
        this.chromosome = chromosome;
    }

    public Individual(int chromosomeLength) {
        this.chromosome = new int[chromosomeLength];
        for (int gene = 0; gene < chromosomeLength; gene++) {
            if (0.5 < Math.random()) {
                this.setGene(gene, 1);
            } else {
                this.setGene(gene, 0);
            }
        }
    }

    public int[] getChromosome() {
        return this.chromosome;
    }
    public int getChromosomeLength() {
        return this.chromosome.length;
    }

    public void setGene(int offset, int gene) {
        this.chromosome[offset] = gene;
    }

    public int getGene(int offset) {
        return this.chromosome[offset];
    }

    public void setFitness(double fitness) {
        this.fitness = fitness;
    }

    public double getFitness() {
        return this.fitness;
    }

    public String toString() {
        String output = "";
        for (int gene = 0; gene < this.chromosome.length; gene++) {
            output += this.chromosome[gene];
        }
        return output;
    }
}
```

Populaton.java :

```
package Lab3;

import java.util.Arrays;
import java.util.Comparator;
import java.util.Random;

public class Population {
    private Individual population[];
    private double populationFitness = -1;

    public Population(int populationSize) {
        this.population = new Individual[populationSize];
    }

    public Population(int populationSize, int chromosomeLength) {
        this.population = new Individual[populationSize];

        for (int individualCount = 0; individualCount < populationSize;
individualCount++) {
            Individual individual = new Individual(chromosomeLength);
            this.population[individualCount] = individual;
        }
    }

    public Individual[] getIndividuals() {
        return this.population;
    }

    public Individual getFittest(int offset) {
        Arrays.sort(this.population, new Comparator<Individual>() {
            @Override
            public int compare(Individual o1, Individual o2) {
                if (o1.getFitness() > o2.getFitness()) {
                    return -1;
                } else if (o1.getFitness() < o2.getFitness()) {
                    return 1;
                }
                return 0;
            }
        });

        return this.population[offset];
    }

    public void setPopulationFitness(double fitness) {
        this.populationFitness = fitness;
    }
    public double getPopulationFitness() {
        return this.populationFitness;
    }
    public int size() {
        return this.population.length;
    }

    public Individual setIndividual(int offset, Individual individual) {
        return population[offset] = individual;
    }

    public Individual getIndividual(int offset) {
        return population[offset];
    }

    public void shuffle() {
        Random rnd = new Random();
        for (int i = population.length - 1; i > 0; i--) {
            int index = rnd.nextInt(i + 1);
            Individual a = population[index];
            population[index] = population[i];
            population[i] = a;
        }
    }
}
```

GeneticAlgorithm.java :

```
 package Lab3;

 public class GeneticAlgorithm {
     private int populationSize;
     private double mutationRate;
     private double crossoverRate;
     private int elitismCount;

     public GeneticAlgorithm(int populationSize, double mutationRate, double crossoverRate,
int elitismCount) {
         this.populationSize = populationSize;
         this.mutationRate = mutationRate;
         this.crossoverRate = crossoverRate;
         this.elitismCount = elitismCount;
     }

    Population population = new Population(this.populationSize, chromosomeLength);
         return population;
     }

     public double calcFitness(Individual individual) {

         int correctGenes = 0;
         for (int geneIndex = 0; geneIndex < individual.getChromosomeLength();
 geneIndex++) {
             if (individual.getGene(geneIndex) == 1) {
                 correctGenes += 1;
             }
         }
         double fitness = (double) correctGenes / individual.getChromosomeLength();
         individual.setFitness(fitness);
         return fitness;
     }

     public void evalPopulation(Population population) {
         double populationFitness = 0;
         for (Individual individual : population.getIndividuals()) {
             populationFitness += calcFitness(individual);
         }
         population.setPopulationFitness(populationFitness);
     }
```

```
        public boolean isTerminationConditionMet(Population population) {
            for (Individual individual : population.getIndividuals()) {
                if (individual.getFitness() == 1) {
                    return true;
                }
            }

            return false;
        }

        public Individual selectParent(Population population) {
            Individual individuals[] = population.getIndividuals();
            // Spin roulette wheel
            double populationFitness = population.getPopulationFitness();
            double rouletteWheelPosition = Math.random() * populationFitness;
            // Find parent
            double spinWheel = 0;
            for (Individual individual : individuals) {
                spinWheel += individual.getFitness();
                if (spinWheel >= rouletteWheelPosition)
                    return individual;
             }
            return individuals[population.size() - 1];
        }

        public Population crossoverPopulation(Population population) {
            Population newPopulation = new Population(population.size());
            for (int populationIndex = 0; populationIndex < population.size();
    populationIndex++) {
                Individual parent1 = population.getFittest(populationIndex);
                if (this.crossoverRate > Math.random() && populationIndex >=
                    this.elitismCount) {

                 Individual offspring = new Individual(parent1.getChromosomeLength(         ));

                    Individual parent2 = selectParent(population);
                    for (int geneIndex = 0; geneIndex < parent1.getChromosomeLength();
geneIndex++) {
                        if (0.5 > Math.random()) {
                            offspring.setGene(geneIndex, parent1.getGene(geneIndex));
                        } else {
                            offspring.setGene(geneIndex, parent2.getGene(geneIndex));
                        }
                    }
                    newPopulation.setIndividual(populationIndex, offspring);
                } else {
                    newPopulation.setIndividual(populationIndex, parent1);
                }
            }

            return newPopulation;
        }

        public Population mutatePopulation(Population population) {
            Population newPopulation = new Population(this.populationSize);
            for (int populationIndex = 0; populationIndex < population.size(); populationIndex++) {
                Individual individual = population.getFittest(populationIndex);
                for (int geneIndex = 0; geneIndex < individual.getChromosomeLength(); geneIndex++)
{
                    // Skip mutation if this is an elite individual
                    if (populationIndex > this.elitismCount) {
                        // Does this gene need mutation?
                        if (this.mutationRate > Math.random()) {
                            // Get new gene
                            int newGene = 1;
                            if (individual.getGene(geneIndex) == 1) {
                                newGene = 0;
                            }
                            // Mutate gene
                            individual.setGene(geneIndex, newGene);
                        }
                    }
                }
                newPopulation.setIndividual(populationIndex, individual);
            }

           return newPopulation;
        }

    }
```

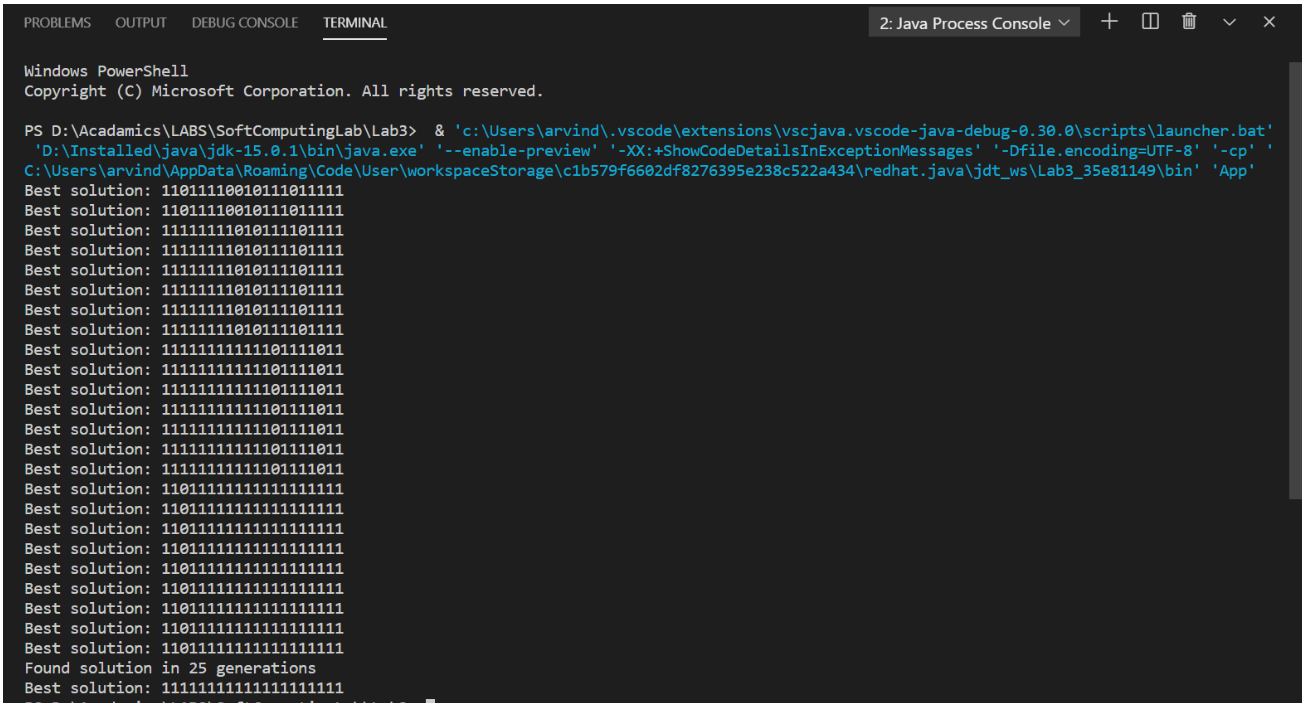
App.java :

```
//import Lab3.Individual;
import Lab3.Population;
import Lab3.GeneticAlgorithm;

public class App {
        private static final int numberOfBits = 20;

        public static void main(String[] args) {
        GeneticAlgorithm ga = new GeneticAlgorithm(100, 0.001, 0.95, 2);
        Population population = ga.initPopulation(numberOfBits);
        ga.evalPopulation(population);
        int generation = 1;
        while (ga.isTerminationConditionMet(population) == false) {
            System.out.println("Best solution: " + population.getFittest(0).toString());
            population = ga.crossoverPopulation(population);
            population = ga.mutatePopulation(population);
            ga.evalPopulation(population);
            generation++;
        }
        System.out.println("Found solution in " + generation + " generations");
        System.out.println("Best solution: " + population.getFittest(0).toString());
    }
}
```

Output :



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS D:\Acadamics\LABS\SoftComputingLab\Lab3>  & 'c:\Users\arvind\.vscode\extensions\vscjava.vscode-java-debug-0.30.0\scripts\launcher.bat'
 'D:\Installed\java\jdk-15.0.1\bin\java.exe' '--enable-preview' '-XX:+ShowCodeDetailsInExceptionMessages' '-Dfile.encoding=UTF-8' '-cp' '
C:\Users\arvind\AppData\Roaming\Code\User\workspaceStorage\c1b579f6602df8276395e238c522a434\redhat.java\jdt_ws\Lab3_35e81149\bin' 'App'
Best solution: 11011110010111011111
Best solution: 11011110010111011111
Best solution: 11111111010111101111
Best solution: 11111111010111101111
Best solution: 11111111010111101111
Best solution: 11111111010111101111
Best solution: 11111111010111101111
Best solution: 11111111010111101111
Best solution: 11111111111101111011
Best solution: 11111111111101111011
Best solution: 11111111111101111011
Best solution: 11111111111101111011
Best solution: 11111111111101111011
Best solution: 11111111111101111011
Best solution: 11111111111101111011
Best solution: 11011111111111111111
Best solution: 11011111111111111111
Best solution: 11011111111111111111
Best solution: 11011111111111111111
Best solution: 11011111111111111111
Best solution: 11011111111111111111
Best solution: 11011111111111111111
Best solution: 11011111111111111111
Best solution: 11011111111111111111
Found solution in 25 generations
Best solution: 11111111111111111111
```