Source code

Individual.java

```
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.ThreadLocalRandom;
/**
 * Created by yauheni on 07.09.16.
 */
class Individual implements Comparable<Individual> {
    public static final int GENE_LENGTH = 1000;
    private List<Integer> genes;
    private int fitnessValue;
    public int getGene(int index) {
        return genes.get(index);
    }
    public void setGene(int index, int gene) {
        genes.set(index, gene);
    }
    public Individual() {
        genes = new ArrayList<>();
        for (int i = 0; i < GENE_LENGTH; ++i) {
            genes.add(ThreadLocalRandom.current().nextInt(0, 2));
        }
    }
    public int getFitnessValue() {
        return fitnessValue;
    }
    public void setFitnessValue(int fitnessValue) {
        this.fitnessValue = fitnessValue;
    }
    public void generateIndividual() {
    }
    public int getFitness() {
        int temp = 0;
        for (int i = 0; i < genes.size(); i++) {
            if(genes.get(i) == 1) {
                temp++;
            }
        }
        fitnessValue = temp;
        return fitnessValue;
    }
    @Override
    public int compareTo(Individual o) {
        return (o.getFitness() > fitnessValue ? 1 : (o.getFitness() ==
fitnessValue) ? 0 : -1);
    }
}
```

Population.java

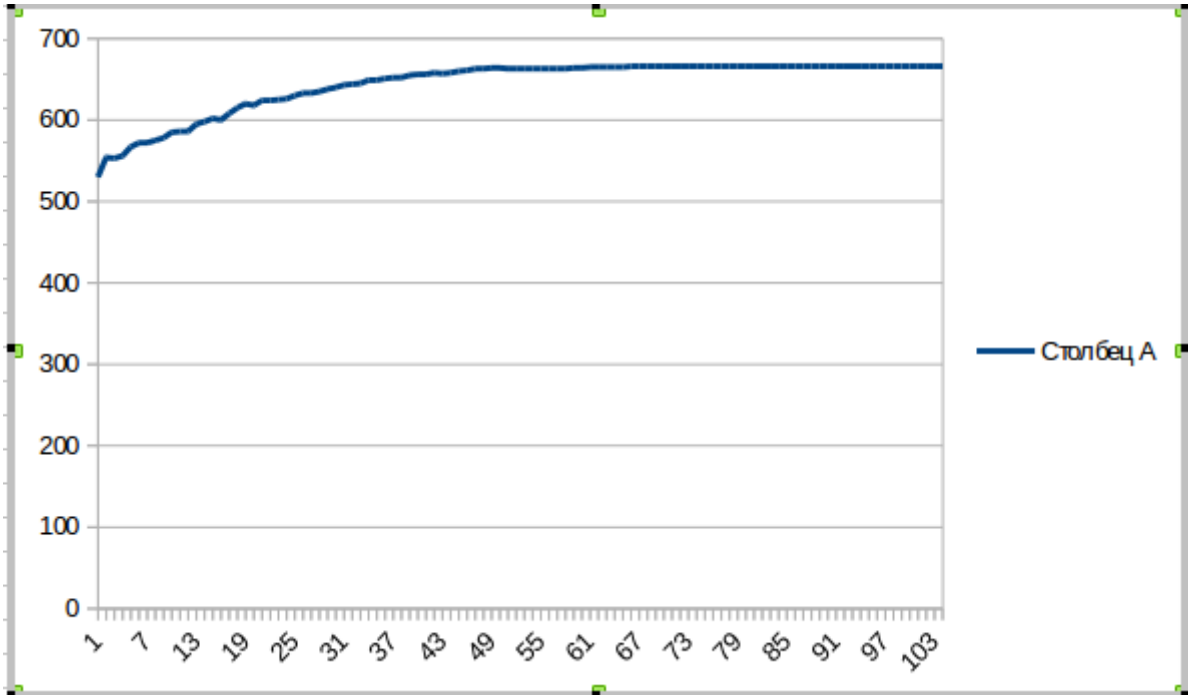
```
import java.util.*;
class Population {
    public static final int POPULATION_SIZE = 100;
    private List<Individual> individuals;
    public Population(List<Individual> individuals) {
        this.individuals = individuals;
    }
    public Population() {
    }
    public static Population newPopulation() {
        Population pop = new Population();
        List<Individual> list = new ArrayList<>();
        for (int i = 0; i < 100; i++) {
            list.add(new Individual());
        }
        pop.setIndividuals(list);
        return pop;
    }
    public List<Individual> getIndividuals() {
        return individuals;
    }
    public void setIndividuals(List<Individual> individuals) {
        this.individuals = individuals;
    }
    public Individual getFittestIndividual() {
        try {
            Collections.sort(individuals);
        }catch(Exception e) {
        }
        return individuals.get(0);
    }
    public double averageFitness() {
        int temp = 0;
        for (int i = 0; i < 100; i++) {
            for (int j = 0; j < 1000; j++) {
                if(individuals.get(i).getGene(j) == 1) {
                    temp++;
                }
            }
        }
        return temp/100;
    }
    public double getMax() {
        int temp = 0;
        Individual best = getFittestIndividual();
        for (int i = 0; i < 1000; i++) {
            if(best.getGene(i) == 1){
                temp++;
            }
        }
        return temp;
    }
}
```

Main.java

```
public class Main {
    public static void main(String[] args) {
        Population pop = Population.newPopulation();
        List<Double> tempList = new ArrayList<>();
        int count=0;
        while(true) {
            if(pop.getMax() == pop.averageFitness())
                count++;
            if(count == 200)
                break;
            System.out.print(pop.getMax() + "\n");
            tempList.add(pop.averageFitness());
            Population newPopulation = new Population();
            List<Individual> newIndividuals = new ArrayList<>();
            List<Individual> individuals = pop.getIndividuals();
            for (int i = 0; i < 50; i++) {
                int first = ThreadLocalRandom.current().nextInt(0, 50);
                int second = ThreadLocalRandom.current().nextInt(0, 50);
                int razrez = ThreadLocalRandom.current().nextInt(0, 1000);
                Individual firstInd = individuals.get(first);
                Individual secondInd = individuals.get(second);
                Individual newFirst = new Individual();
                Individual newSecond = new Individual();
                for (int j = 0; j < 1000; j++) {
                    if(j < razrez) {
                        newFirst.setGene(j, firstInd.getGene(j));
                        newSecond.setGene(j, secondInd.getGene(j));
                    } else {
                        newFirst.setGene(j, secondInd.getGene(j));
                        newSecond.setGene(j, firstInd.getGene(j));
                    }
                }
                newIndividuals.add(newFirst);
                newIndividuals.add(newSecond);
            }
            newPopulation.setIndividuals(newIndividuals);
            pop = newPopulation;
        }
        System.out.println("============================");
        for (int i = 0; i < tempList.size(); i++) {
            System.out.println(tempList.get(i));
        }
    }
}
```

After compiling the code, we have got next result — 66,6% good gens. Average value is the same.

Graph for best chromosomes:



Graph for average values from generation: