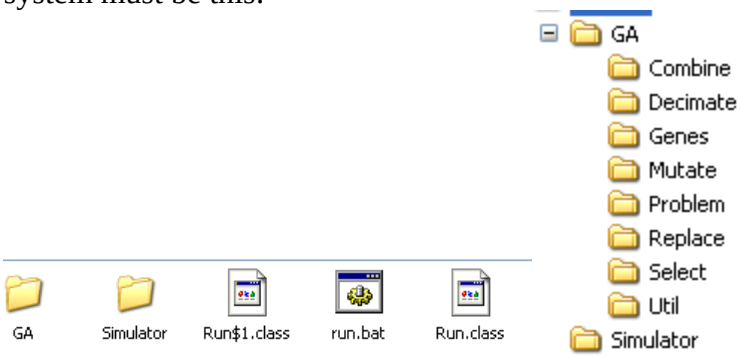
MINI TUTORIAL for MUGA

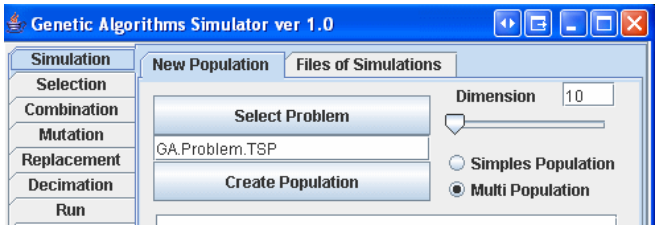
MUGA Simulator is a dynamic loader for problems and genetic methods. To run the file system must be this:



Decompress de zip file and execute the run.bat.

MUGA has a intuitive interface to make simulations in genetic Algoritms. To do one simulation just fallows the steps 1 to 7.

1 – Select the problem and create population



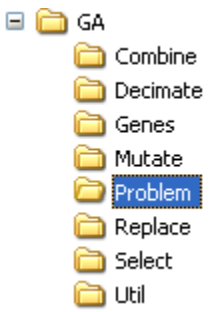
In this screen user select the problem and create the population. The dimension of population is selected int the slider or text box.
Population could be **Simples** or **Multi** . Multi populations take advantages in some methods.

1.1 make yours problems

User can create news problems and solve them in simulator.
To create a new problem write a java class, compile them and **put the *.class in the problem directory**.
We have two types of problems:

- Numerical optimization (AbstractIndividual)
- Combinatorial optimizations (AbstractPermutationIndividual).

```
package GA.Problem;
import GA.Genes.GeneBinary;
import GA.Problem.AbstractIndividual;
public class Simples extends AbstractIndividual{
    public Simples() {
         super();
        this.addGene( new GeneBinary(4));
        this.addGene( new GeneBinary(4));
    }
 public double Fitness() {
        int v1 = (int) genome.getValue(0);
        int v2 = (int) genome.getValue(1);
        return v1 + v2*16;
    }
```



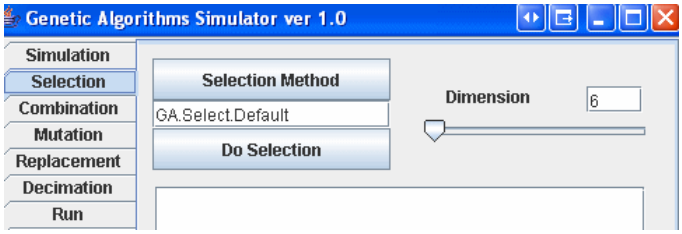
Or

```
package GA.Problem;
import GA.Genes.GeneInteger;
public class SimplesPermutation extends AbstractPermutationIndividual{
    public SimplesPermutation() {
        this.genome.addGene( new GeneInteger(0,4, 0) ); //min max value
        this.genome.addGene( new GeneInteger(0,4, 1) );
        this.genome.addGene( new GeneInteger(0,4, 2) );
        this.genome.addGene( new GeneInteger(0,4, 3) );
        // GENE  [ 0 1 2 3 ]
    }
    public double Fitness() {
        // simples function   [ g0 + g1^2 +  g2^3 + g3^4 ]
        // [0, 1, 2, 3] = 90.0
        int v1 = (int) genome.getValue(0);
        int v2 = (int) genome.getValue(1);
        int v3 = (int) genome.getValue(2);
        int v4 = (int) genome.getValue(3);

        return v1 + java.lang.Math.pow(v2,2)
                  + java.lang.Math.pow(v3,3)
                  + java.lang.Math.pow(v4,4);
    }
}
```

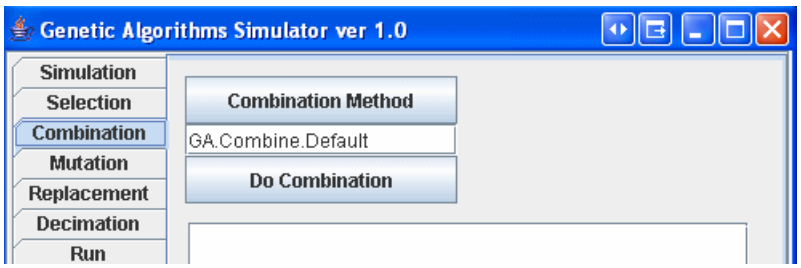
The user could save or load entire simulations.

2 – Selection



Select and execute the method and the number of parents in population to reproduction.

3 - Combination

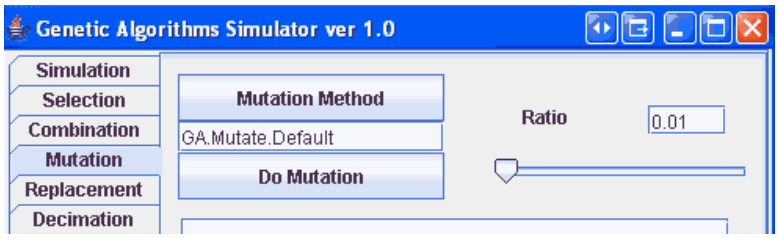


Select and execute the combination method in selected parents.
If the problem is derived from AbstractPermutationIndividual the methods for combination are:

- MultiPMXCrossover
- PMXCrossover
- OXCrossover
- UXCrossover

The others is for the AbstractIndividual

4 - Mutation

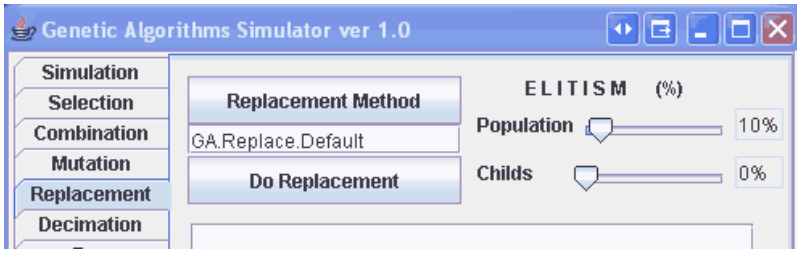


Selects and Execute de mutation method and the mutation ratio.
If the problem is derived from AbstractPermutationIndividual the methods for combination are:

- SwapGenes

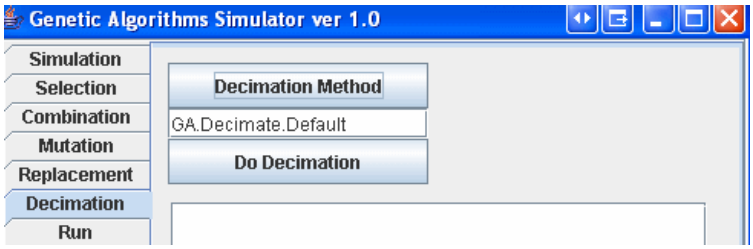
The others is for the AbstractIndividual

5 - Replacement



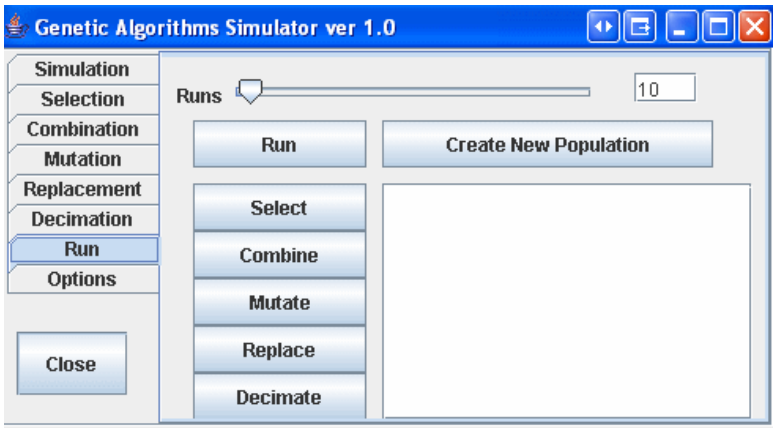
Select and execute de replacement of the parents for the childrens. Select the percentage of the elitism in population and childs.

6 - Decimation



Special operator to multipopulations.

7 – Run



Execute the selection, combination, mutation, replacement and decimation in population, or all in the Run button.