# CSE443 – Object Oriented Analysis and Design – Homework 2

161044042 – Yasir Nacak

## Q1.

To solve this problem, I first created a base class that can do what a genetic algorithm should basically be doing. In order to do that, I created Select, Crossover and Mutate methods. I then added the limitations of this simulation as class field variables. These variables are the maximum population in each generation, percentage of mutation for each gene in each chromosome of each newly born children, maximum value between worst and best fit chromosomes for simulation to stop (to detect convergence) and finally the number of crossover points when genes are passed from parents to children.

I also created a Chromosome class that represents each X1 and X2 variables in our space. This Chromosome automatically calculates its fitness upon creation. The fitness value defined is the result that our equation gives plus the minimum value that our equation can take (which is -75 in this problem). This way, every chromosome in our current generation has positive fitness values from 0 to around 200.

In my Select method, I left it as abstract for child classes to override using different parent chromosome selection methods. This way I made that method a Template Method. Then for Crossover, I converted X1 and X2 variables of each parent to their binary string representation and picked substrings based on crossover points and added them to each other so I can get a mixture of both parent's genes. And finally, in my Mutate method, I went through the binary string representation of each children and randomly flipped bits based on my mutation percentage variable. This way I avoided early convergences in my generations.

For each of my child classes that derive from the previously described base class, I override the Select method based on a different parent chromosome selection method. These selection methods are:

1. Roulette Wheel Selection
2. Rank Selection
3. Tournament Selection

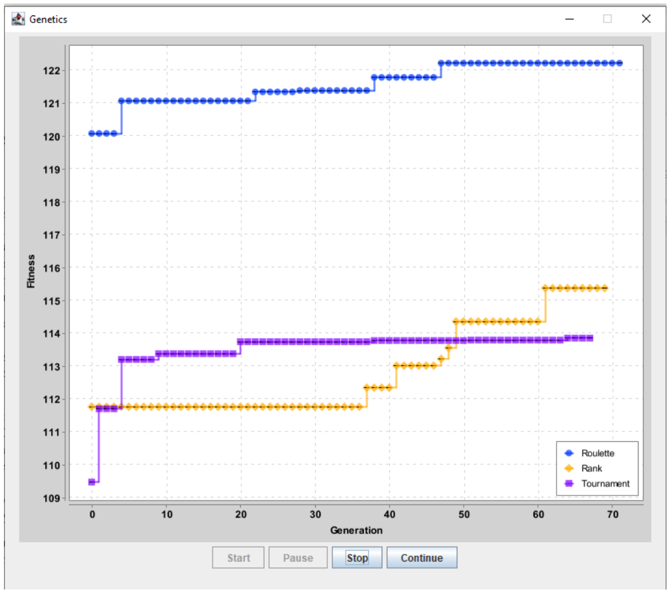
For Roulette Wheel Selection, I calculated the sum of each fitness value in my current population and generated a number between 0 and the sum I calculated. I then started summing up fitness values again in a partial sum. Whenever I reached the generated value on my partial sum, I stopped and chose the current chromosome that I was adding fitness value from.

For Rank Selection, I gave values from 0 to current population to each chromosome in my population and I repeated the partial sum method from the Roulette Wheel Selection.

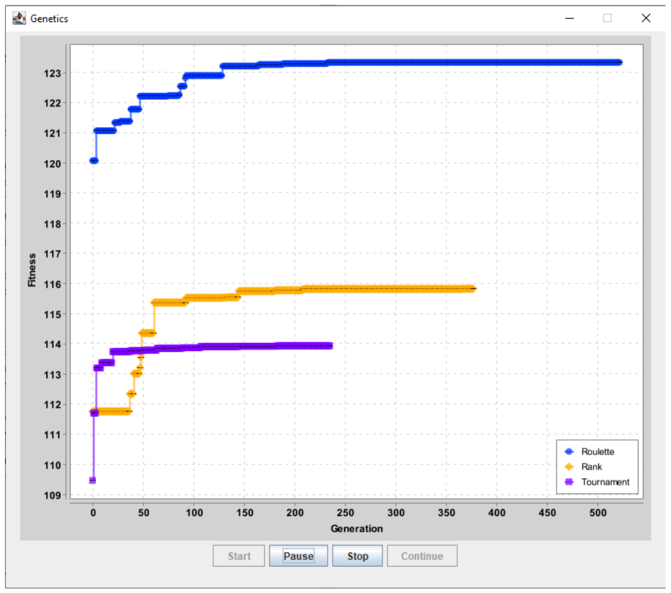
For Tournament Selection, I picked quarter of my population randomly and picked the fittest one from them to get a parent

## Q2.

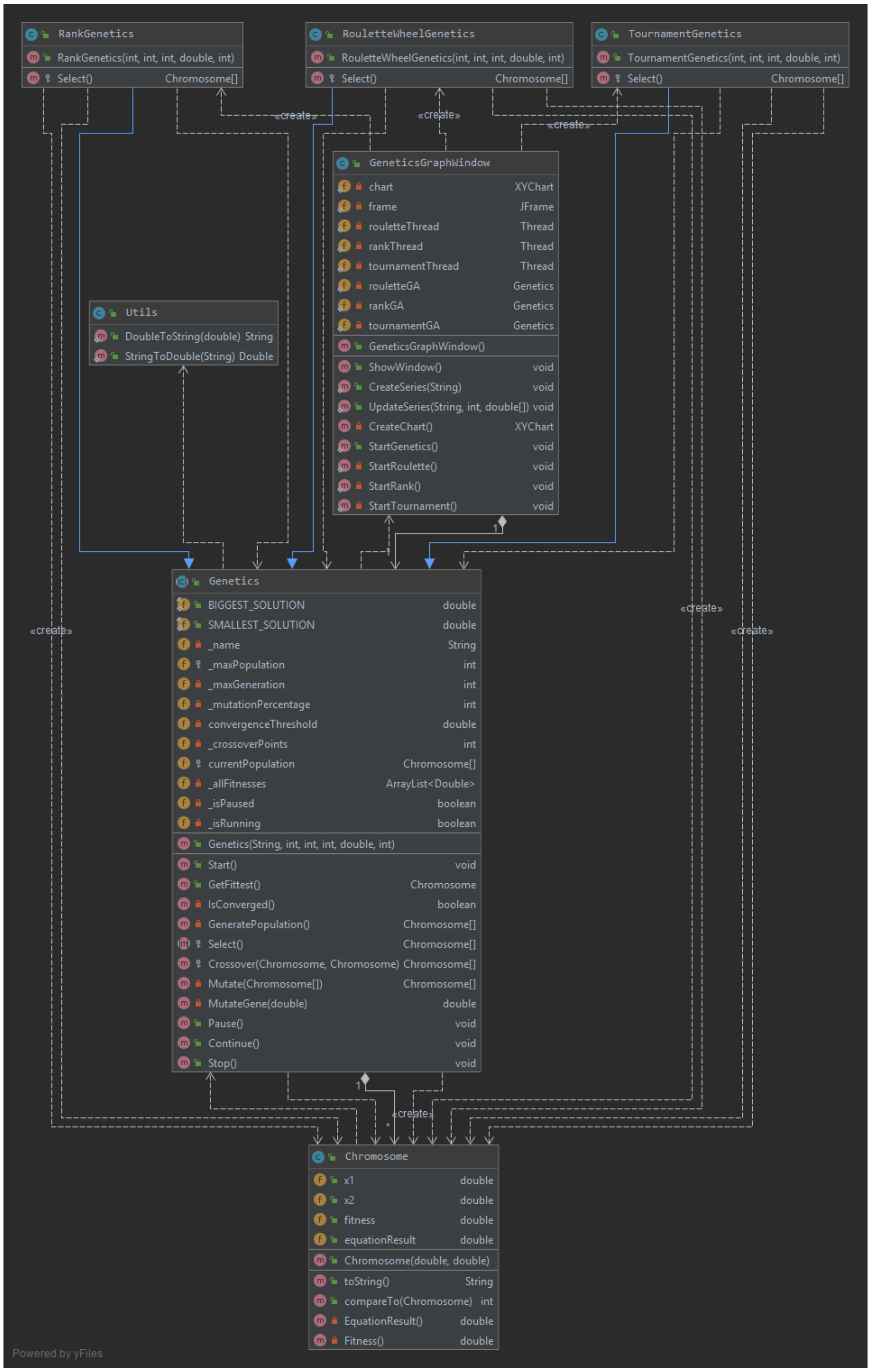
To do this part, I utilized a third-party library called XChart to display a chart on a JFrame window. In this window, I refreshed my chart after each iteration of each genetic algorithm. I did this by updating the window from my genetic algorithms. And to make all three of my algorithms work in parallel, I created a thread for each of them and started them right after the other. This way, I displayed generations of my genetic algorithms in real time. I also added buttons for used to pause / continue the simulation and stop and restart it as well.



Around 70th generations of each algorithm



Same runtime after all algorithms converge



Class diagram of the whole system also including windowing