**Goal:**

The goal of this Project is to write a genetic algorithm that solves the 01 Knapsack Problem.

Briefly stated, the Knapsack Problem is: You have a collection of N objects of different weights, w1, w2, …, wn, and different values, v1, v2, …, vn, and a knapsack that can only hold a certain maximum combined weight W. You would like to get a set of objects of maximal value into the knapsack.

**Implementation Design**

*Genetic code:* Each base is an integer corresponding to an Item. Each gene is an array of integers which are taken to filled the capacity of the knapsack

*Gene expression:* The individual genes code for particular traits.

*Fitness function:* Fitness is checked for each new population. I am evaluating for the maximum item which can fit into the knapsack. For Fitness, we are taking two traits, volume and weight. If the bit in the gene is 1 then the weight and volume of the item in that index is added to the total capacity and total volume respectively. If the total capacity and volume is less than or equal to the sack capacity and sack volume then that fitness value is returned. The population with the highest fitness rate is selected as the best in that generation.

*Mutation:* I am doing Bit Flip mutations of randomly selected bases. So, when the randomly selected bit has 1 it is changed 0 or if the bit has a 0, it is changed as 1. For example:

Parent: [0, 1, 1, 0, 1, 0, 1, 1, 1, 1]

Bit Flip index 5

Child: [0, 1, 1, 0, 0, 0, 1, 1, 1, 1]

*Crossing Over:*

If the probability is less the 0.5 then crossover functionality is performed per the crossover point.

The crossover point is randomly generated.

For example, the crossover point is at index 5:

Parent 1: [0, 1, 0, 0, 0, 0, 1, 1, 1, 1]

Parent 2: [0, 1, 1, 0, 1, 0, 0, 1, 0, 1]

Child: [0, 1, 0, 0, 0, 0, 0, 1, 0, 1] ↑

In Parent 1 we find the index of the value that has index 5, and in Parent2 after index 5. We take the value from start till index 5 from parent1 and the values after index 5 till end.

Evolution:

The item are generated by randomgeneratedItems() method and the

the population of generation of gen0 is by randomgeneratedPopulation() using generatingGene() method.

I am using sexual reproduction(crossover) and asexual reproduction(mutation) to generate variation in the new population.

While only using sexual reproduction(crossover) or asexual reproduction(mutation) doesn’t create a good results and stop evolving in the local maximum.

Observations:

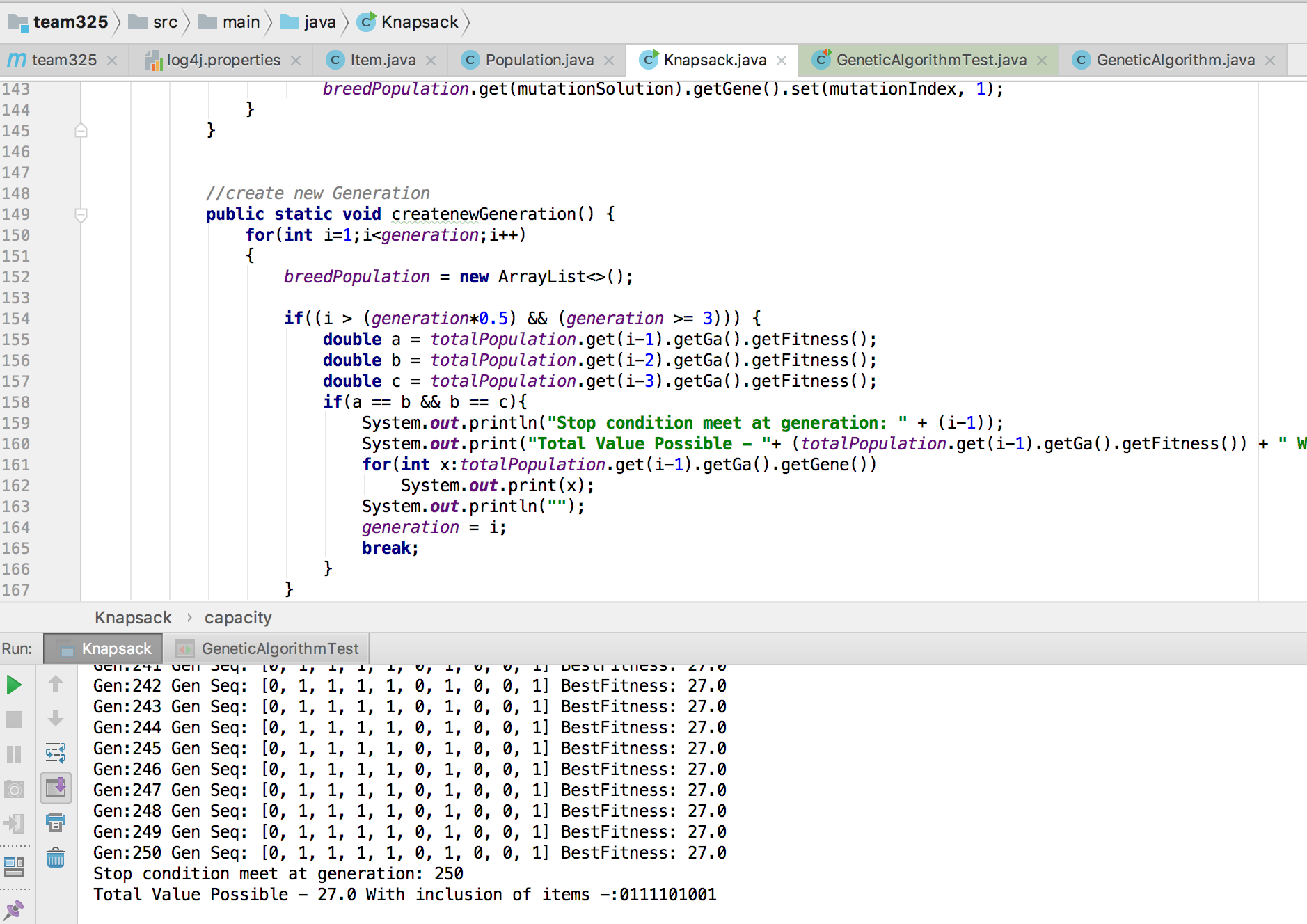
When we use sexual reproduction(crossover) and asexual reproduction(mutation), create a better set of result with better fitness.

When the population size and generation limit was increased the mean fitness of the generation also increased.

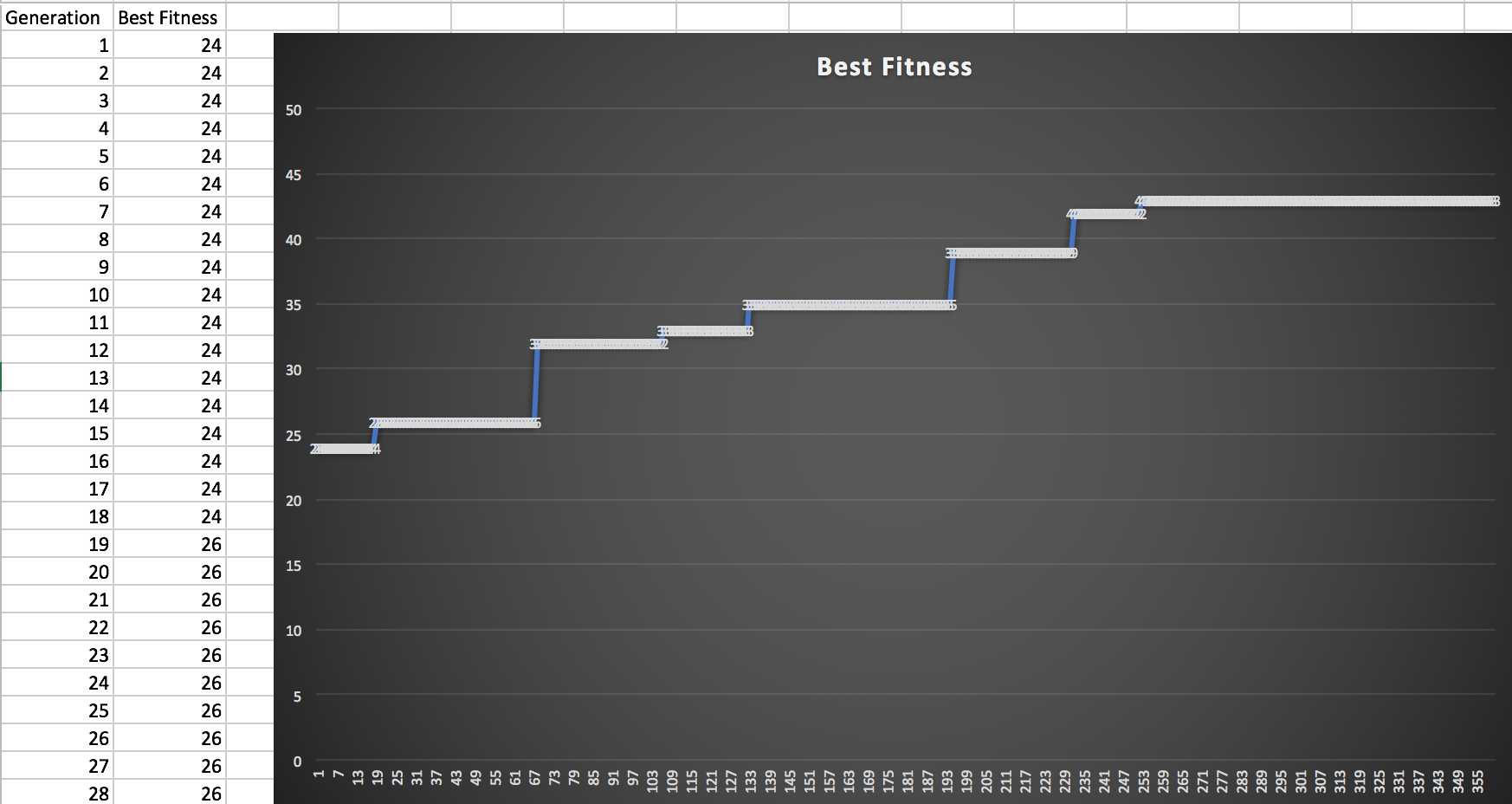
For each generation, after the culling is complete the number of population which culled is logged along with their fitness score. The evolutionary process terminates after 360 generations and when the highest fitness score hasn't changed in 3 generations

Logging is done using log4j.properties and the log is generate in .log.out file in the default package.

Project gives us the result:



Generation vs Best Fitness



Junit Test case execution:

The project passes the test suite.