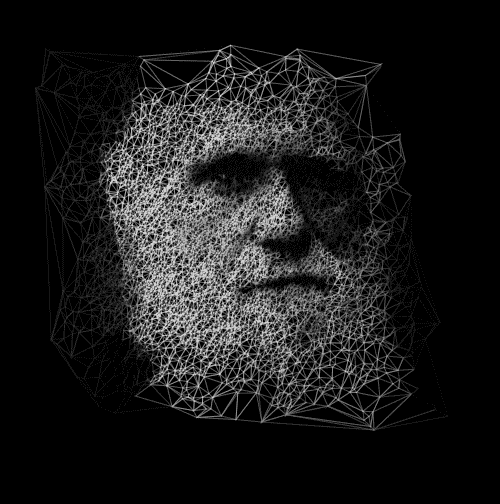
**Evolution Image Generation**

*A process portrayed using Genetic Algorithm.*



**Rajat Jaswal & Srishti Saboo**

Group 219

**2018 Fall**

**INFO6205 Final Project Report**

**Problem Description**

Human evolution is the lengthy process of change by which people originated from apelike ancestors.

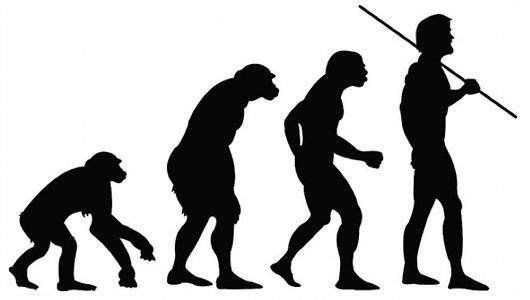
Whenever there was mutation between the Creatures, the gene having the better fitness transfers in the offspring. The process took six million years to transform the creature like Australopithecus, into the creature like a Homo Sapiens.

After these many years, we know the complete transformation of human evolution. We know about the result we are expecting from the problem. So,

the result which we expect is of fitness 100%.

We have the result (the mutation image of fitness 100%), we are starting the process of developing the original image through random set of characters/symbols of random colors and random fonts.

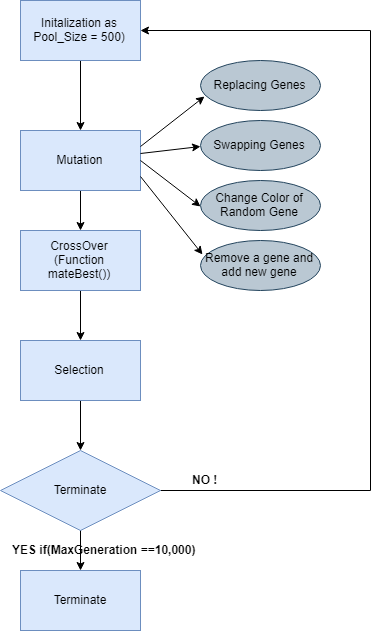
The original image is displayed below.

****

**Genetic Algorithm in Action!**

Genetic Evolution (GE) is a method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. It is a population based, stochastic function minimizer. It involves the following steps-

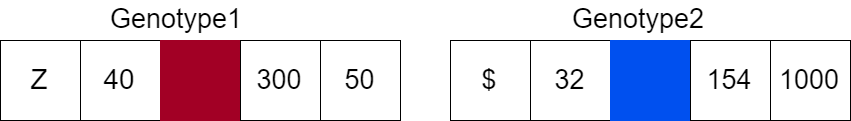
We took initial population of random 1800 (char\_ count) characters as 500 (pool\_size). And the objective function to be maximized was similarity of images. When the value of similarity of parent image and produced image is greater than 0.93, or when the generation count (MAX\_GENERATIONS) crosses 10,000 the algorithm terminates. The basic flow of the algorithm is:

­­

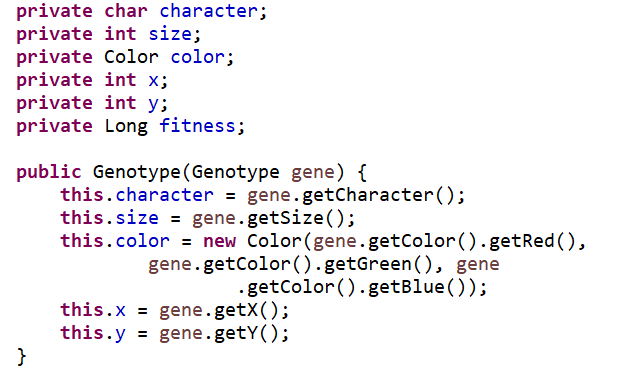
**The Implementation Facet!**

1. **Individual:**

An individual is the image consisting of thousands of random characters with different shapes and colors.  
An individual organism consists of a genetic constitution that is a single character image or a ‘Genotype’ which has a constant encoding type. We represent that as below.

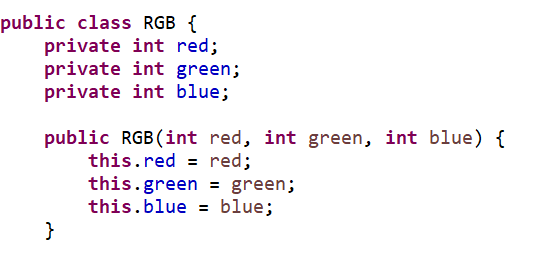


The Genotype is:



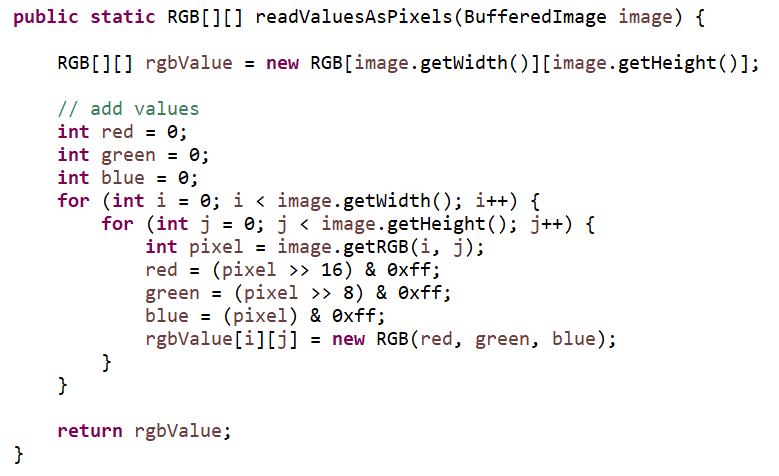
1. The Helper Classes:
2. RGB.java:

It represents the color code in RGB format. It is used to calculate the fitness in pixels.

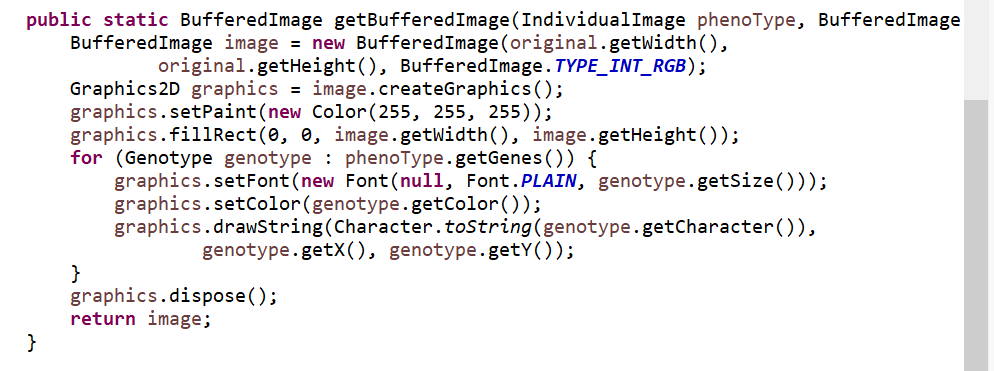


1. ProcessImage.java:

Read the image in the form of RGB matrix which is used to perform calculations based on pixels for further fitness.

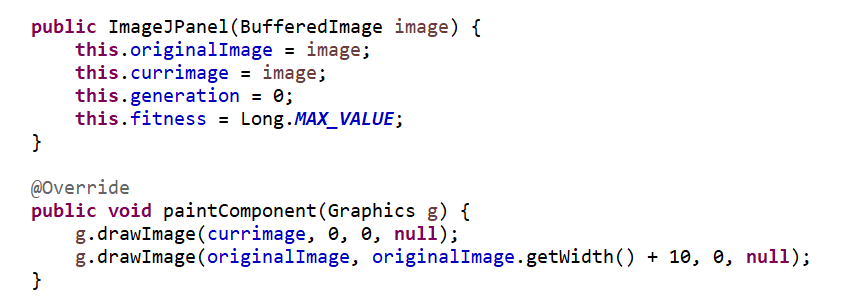


The below method in ProcessImage.java is converting the list of genotypes or an Individual into a BufferedImage which helps us to further visualize the algorithm better.



1. ImageJPanel.java:

ImageJPanel is the interface to visualize the fittest individual after every generation.



1. Main Classes:
2. Main.java:

* The program starts from the main.java class where firstly we read an image and then converted the image into pixels.
* We initialized the population of 500 individuals of 1800 random characters, colors, size, and position each.
* We distributed the fitness evaluator according to the thread, each thread is computing fitness of different set of individuals each.
* During computing the fitness of individuals, we directly compute the fitness of each genotype.
* After finding about the fitness, we get the best fit individual and converted it into an image and display it through the JPanel.
* Until we reach our terminating condition i.e. fitness approximately 0.93, many generations are produced with fitness better than the previous one.

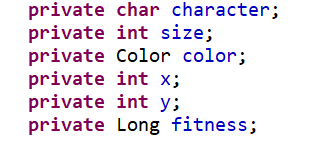
1. Populate.java:

It generates a population of random 1800 characters as 500 individuals. The image returned by population function is of better fitness.



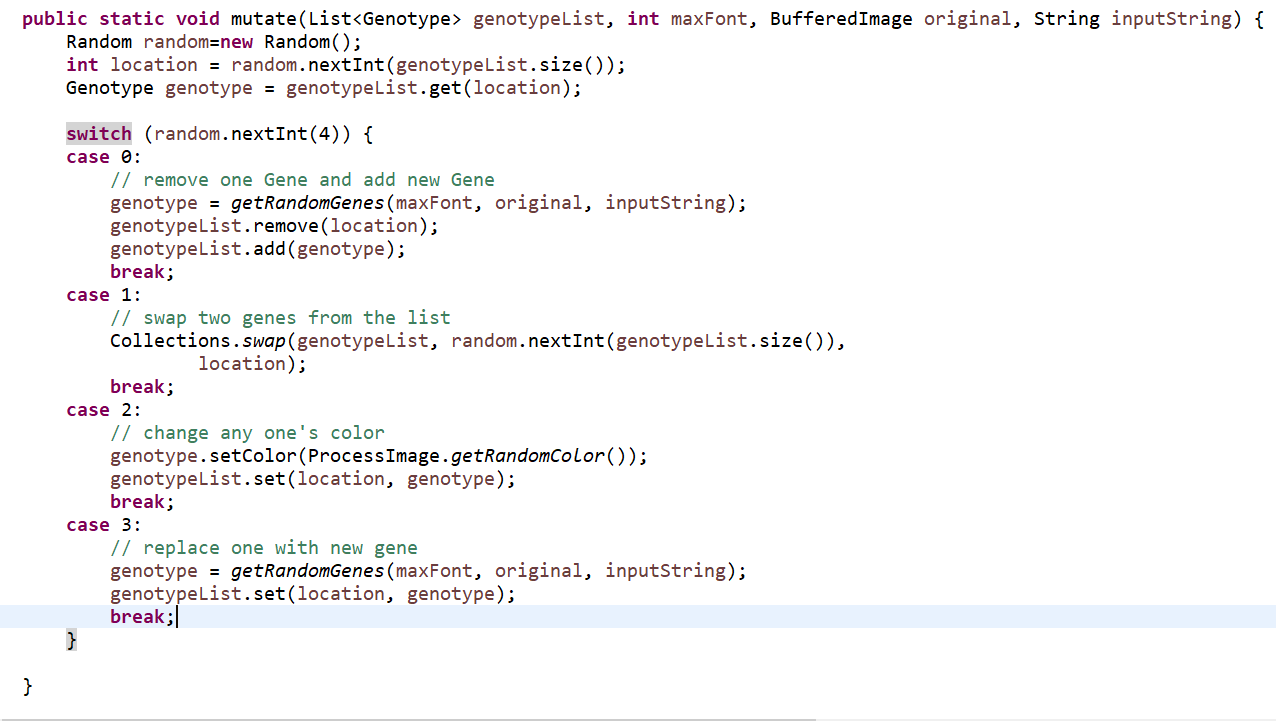
1. Genotype.java:

The genes with different characters, size, color, x-coordinate and y-coordinate are created in this class. The gene level fitness of the respective gene is also stored.

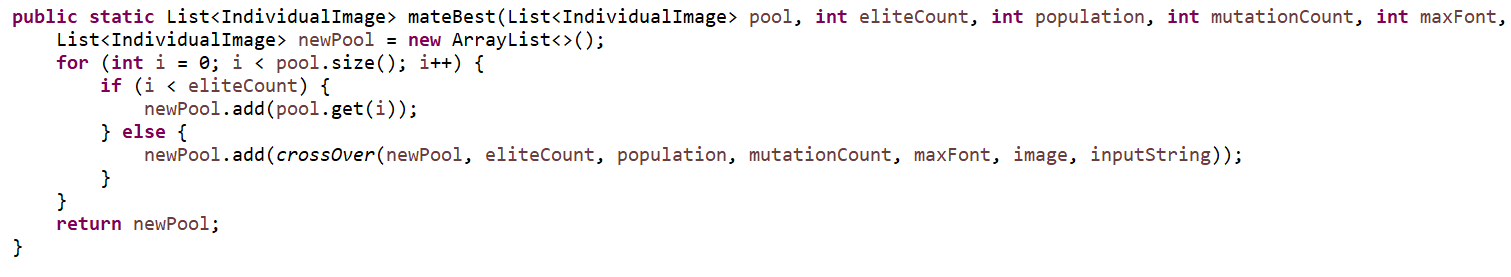


1. Mutation.java:

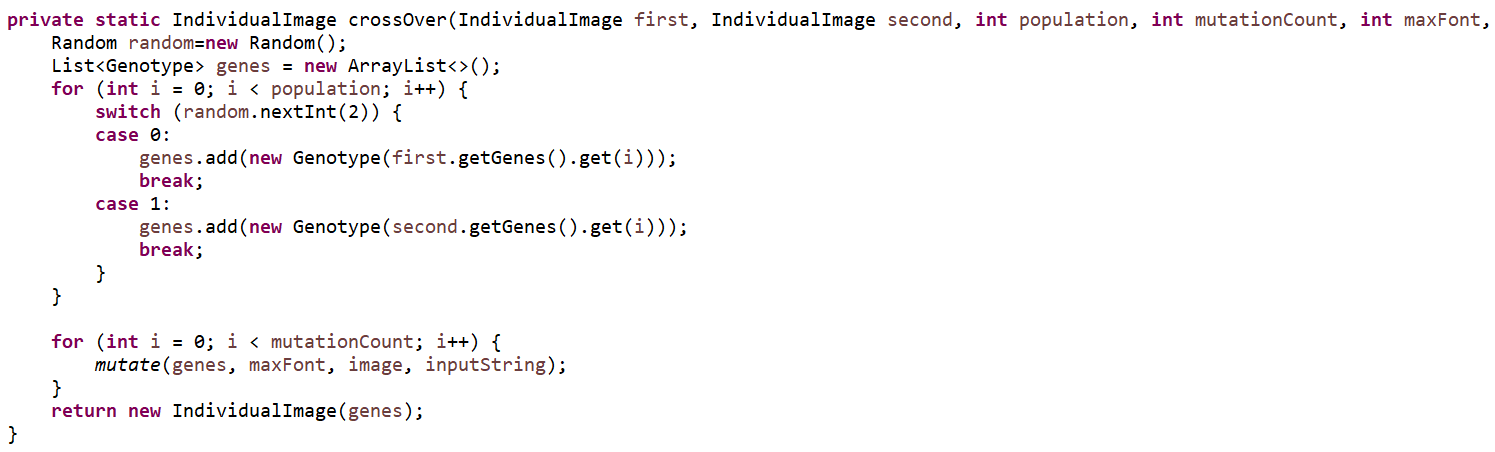
In mutation.java we are having list of gene which we are changing by performing few cases. These cases are like replacing a gene by another gene, swapping two genes from the list or changing any traits of the gene. By performing these cases are goal is to get a gene with the best fitness which provides the best image.



We are performing Elitism for doing crossover, so that always the fittest generation breed in order to give us the best result. We have taken a pool of 10 elite individuals and then breeding through the elites.



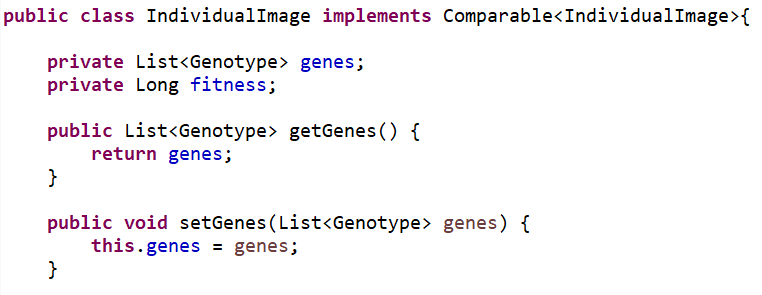
In the Crossover function defined in Mutation.java we are generating one child by randomly taking the traits of any of the two-elite parent.



1. FitnessFunction.java:

In this file we find out the difference of the RGB values of the original image and the image we just generated through randomGene selection. This fitness tells us how far the image is left for it to be breaded more.

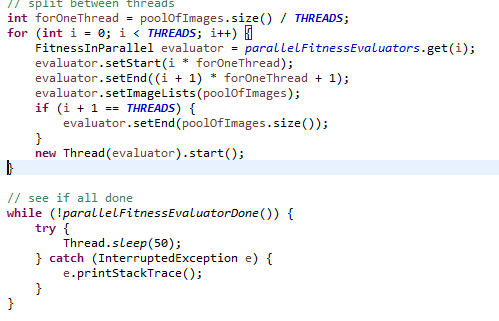


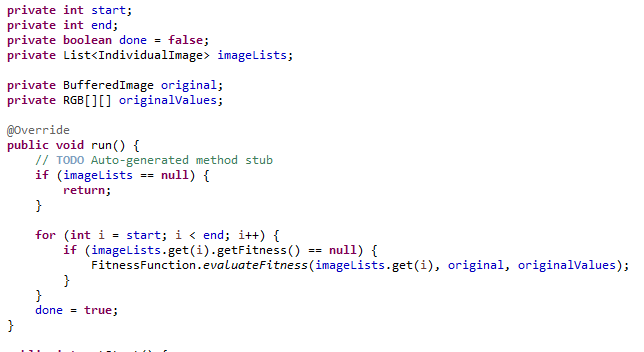
1. IndividualImage.java:  
   An IndividualImage file comprises of a collection of genes and a fitness of its own. This is further used to create a visible interface and is used to mutate and breed further.  
   

**Multithreading, the Cure!**

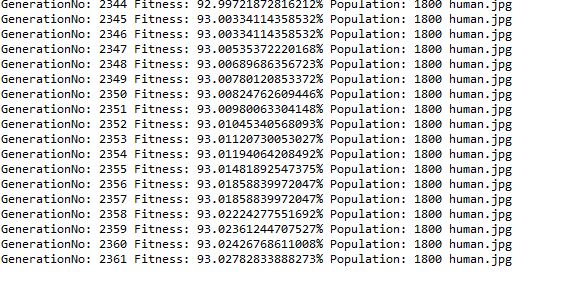
Any huge task in this universe needs to be broken down into smaller tasks so that it takes less time to perform that huge task. We came across such similar problem while calculating fitness for hundreds and thousands of entities.

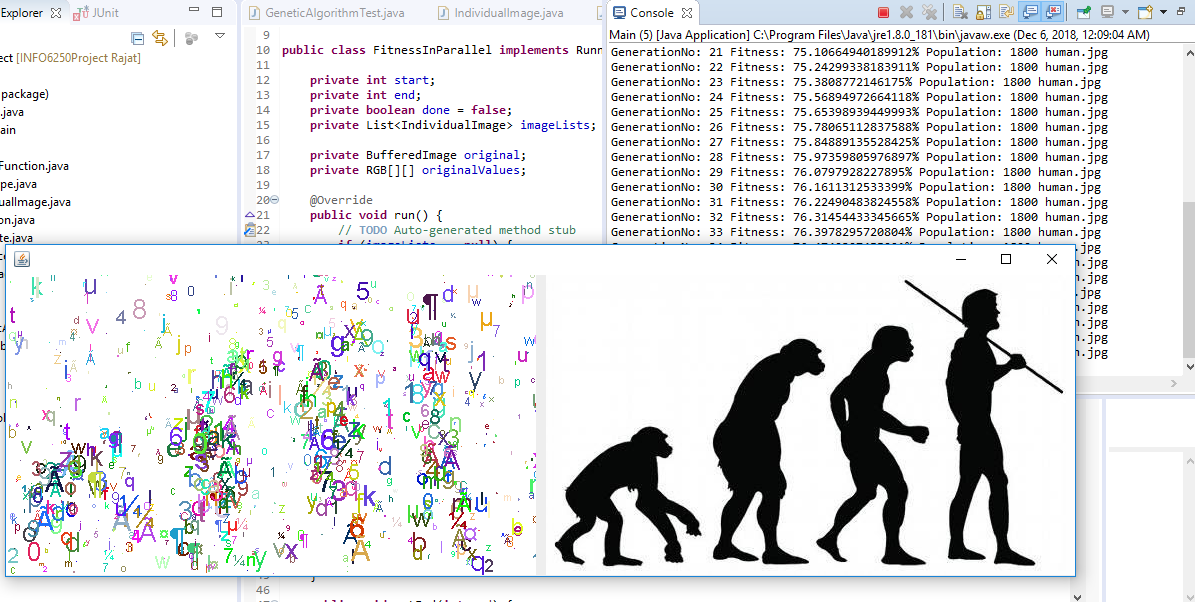
We solved this problem by the concept of multitasking where we initialized some threads and gave a thread each responsibility of solving a problem of multiple entities. This massively impacted on the run time and helped us reducing the overall time of reaching the optimal image.  
  
fitnessofMultipleImages()



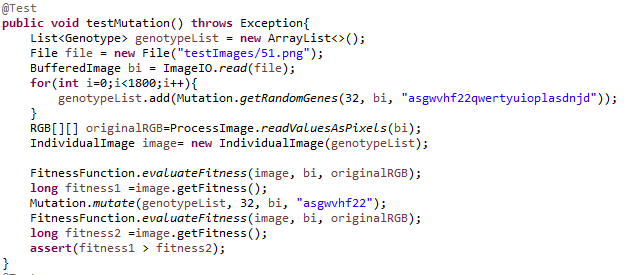
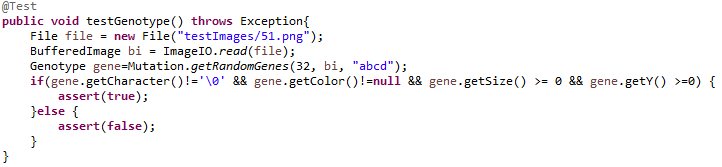
FitnessInParallel.java  


**Output**



Overall Output  


**TEST CASES**

* Test Fitness  
  
* TestMutation  
  
* TestPopulation  
  
* TestGenotype  
  

**GRAPH**