## Genetic Algorithm: (@author-punith thota(18250669))

This is the CS613 project designed as a part of the continuous assessment.

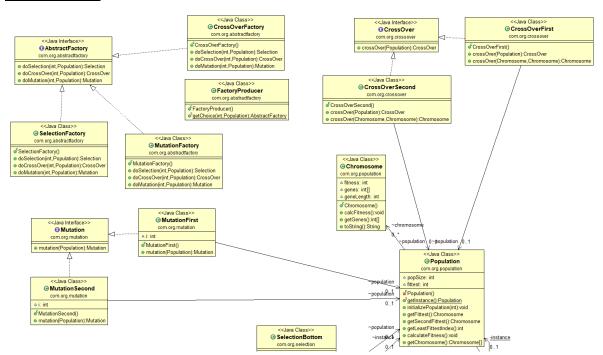
This is a prototype of the Genetic Algorithm but not the actual algorithm. All the functionalities of a Genetic Algorithm are designed with different design patterns as per the requirements.

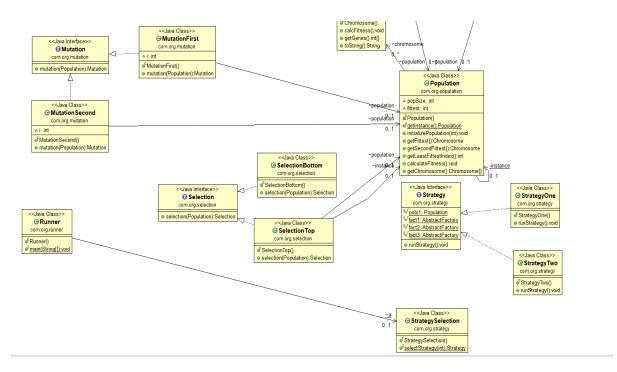
A Genetic Algorithm is basically an evolution algorithm that evolves from an initial population and finally generates the fittest individual.

This algorithm can be used in solving different problems such as the travelling sales person problem.

I designed the prototype of Genetic Algorithm that implements the functionality of Genetic Algorithm such as selection, cross over, mutation and so on...

### Class Diagram:





There are 2 types of selection, cross over, mutation .. all created as sperate classes with separate interfaces respectively.

Two Strategies are implemented for the type of selection, cross over, mutation choices to perform.

#### Singleton Pattern:

Singleton Pattern is the design pattern that lets us to only have one instance of a class

In the code, singleton pattern is implemented in the Population class by making its constructor private and creating a Population type method called get instance that creates only one instance of the Population class and returns the instance whenever called.

## **Code Snippet:**

```
// TODO: Auto-generated Javadoc
/**
 * The Class Population.This is the population class that creates an initial population and
does other functionalities.
 * Singleton pattern is implemented in this class.
 */
public class Population {
    /** The pop size. */
    int popSize = 10;
/** The chromosome. */
```

```
Chromosome[] chromosome = new Chromosome[10];
  /** The fittest. */
  int fittest = 0;
  /** The instance. */
  private static Population instance;
         * Instantiates a new population.
         * The constructor is made private to restrict calling the new method() that is
creating more than on instance of the Population class
         */
        private Population() {
System.out.println("Population Private constructor");
        /**
        * Gets the single instance of Population.
        *This method is used to get the instance of population or create the instance only
once
        * @return single instance of Population
       public static Population getInstance(){
           if(instance == null){
              instance = new Population();
           }
           return instance;
         }
```

#### Abstract Factory:

Abstract Factory is an abstract class or interface that generates products.....In its structure there is a factory producer which produces factories and concrete classes that implement Abstract Factory (SelectionFactory, CrossoverFactory, MutationFactory).

SelectionFactory calls different types of selection.

CrossOverFactory calls different types of crossover.

MutationFactory calls different types of Mutation.

### **Code Snippet:**

package com.org.abstractfactory;

```
import com.org.crossover.CrossOver;
import com.org.mutation.Mutation;
import com.org.population.Population;
import com.org.selection.Selection;
// TODO: Auto-generated Javadoc
* A factory for creating Abstract objects. This is the abstract factory that generates products
*/
public interface AbstractFactory {
       /** Do selection depending on the choice of selection and selection is made on the
population class.
       * @param schoice the schoice
       * @param population the population
       * @return the selection
       */
       Selection doSelection(int schoice, Population population);
       /**
       * Do cross over depending on the choice of crossover and crossover is made on the
selected individuals
       * @param crchoice the crchoice
       * @param population the population
       * @return the cross over
       */
       CrossOver doCrossOver(int crchoice, Population population);
       /**
```

```
* Do mutation with a random probability of doing mutation
        * @param mchoice the mchoice
        * @param population the population
        * @return the mutation
        */
       Mutation doMutation(int mchoice, Population population);
}
Strategy Pattern:
Strategy pattern is a design pattern that allows the strategy to change at run time as per
the user's choice.
For this, different classes are created i.e StrategySelction(client interacts with this
class), Strategy interface and two concrete Strategy classes that implements Strategy
interface.
Code Snippet:
package com.org.strategy;
import com.org.abstractfactory.AbstractFactory;
import com.org.abstractfactory.FactoryProducer;
import com.org.population.Population;
// TODO: Auto-generated Javadoc
* The Interface Strategy creates the variables and declares the strategy method which will
be implemented in the sub classes
*/
public interface Strategy {
```

/\*\* The pobj 1. \*/

Population pobj1= Population.getInstance();

```
//Population pobj2 = new Population();

/** The fact 1. */

AbstractFactory fact1 = FactoryProducer.getChoice(1,pobj1);

/** The fact 2. */

AbstractFactory fact2= FactoryProducer.getChoice(2,pobj1);

/** The fact 3. */

AbstractFactory fact3 = FactoryProducer.getChoice(3,pobj1);

/**

* Run strategy.The method to run different strategies based on user i/p ..this method is implemented in the sub classes.

*/

public void runStrategy();
}
```

# **Output Screenshots:**

```
🔐 Problems 🎯 Javadoc 🖳 Declaration 🗏 Console 🛭
<terminated> Runner (13) [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (15 Dec 2018, 13:08:16)
Enter which strategy to run
1- for SelectionTop,CrossOverFirst,MutationFirst
2- for SelectionBottom, CrossOverSecond, MutationSecond
Population Private constructor
11111
01000
10001
10111
11110
11010
00011
11110
10001
00111
schoice 1
selection top performed
Fittest:[0, 0, 1, 1, 1]
Second Fittest:[1, 1, 1, 1, 1]
crossover 1
Crossover first
Before crossover:
[0, 0, 1, 1, 1]
[1, 1, 1, 1, 1]
After crossover:
[1, 0, 1, 0, 1]
[0, 1, 0, 1, 0]
mutation 1
Mutation first
Mutation happening at random gene
Before mutation [1, 0, 1, 0, 1]
After mutation at position 1 of gene. Changing value to 0 = [1, 0, 1, 0, 1]
Proof of Singleton Class created
object 1:com.org.population.Population@3d4eac69object 2:com.org.population.Population@3d4eac69
```

```
Problems @ Javadoc  □ Declaration □ Console 
<terminated> Runner (13) [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (15 Dec 2018, 13:09:14)
Enter which strategy to run
1- for SelectionTop,CrossOverFirst,MutationFirst
\hbox{2- for SelectionBottom,} CrossOverSecond, \verb+MutationSecond+
Population Private constructor
01000
00110
01000
00000
01111
10110
11100
10100
00011
11111
schoice 2
selection bottom performed
Fittest:[0, 1, 0, 0, 0]
Second Fittest:[1, 1, 1, 1, 1]
crossover 2
Crossover second
Before crossover:
[1, 1, 1, 1, 1]
[0, 1, 0, 0, 0]
After crossover:
[1, 1, 0, 1, 0]
[0, 0, 1, 0, 1]
mutation 2
Mutation second
Mutation happening at random gene
Before mutation [1, 1, 0, 1, 0]
After mutation at position 4 of gene. Changing value to 1 = [1, 1, 0, 1, 1]
Proof of Singleton Class created
object 1:com.org.population.Population@3d4eac69object 2:com.org.population.Population@3d4eac69
```

Data Abstraction is maintained all over with polymorphism(reuse code).

Client only can interact with the Runner and StrategySelection Classes(client interface).

Proof for singleton pattern is also implemented in the Runner class.

# References:

- Design Patterns: <a href="https://www.javatpoint.com/design-patterns-in-java(javatpoint">https://www.javatpoint.com/design-patterns-in-java(javatpoint)</a>
- Design Patterns: <a href="https://www.tutorialspoint.com/design-pattern/(tutorialspoint">https://www.tutorialspoint.com/design-pattern/(tutorialspoint)</a>
- Design Patterns:Head First Design Patterns (pdf)-for conceptual examples.
- Abstract Factory: https://www.tutorialspoint.com/design\_pattern/abstract\_factory\_pattern.htm
- Singleton Pattern: https://www.tutorialspoint.com/design\_pattern/singleton\_pattern.htm
- Strategy Pattern: https://www.tutorialspoint.com/design\_pattern/strategy\_pattern.htm
- Strategy Pattern: <a href="https://www.journaldev.com/1754/strategy-design-pattern-in-java-example-tutorial">https://www.journaldev.com/1754/strategy-design-pattern-in-java-example-tutorial</a>
- Genetic Algorithm: Code snippets for Population, chromosome logic <u>https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3</u>
- Genetic Algorithm Basic Concept: <a href="https://en.wikipedia.org/wiki/Genetic algorithm">https://en.wikipedia.org/wiki/Genetic algorithm</a>
- Genetic Algorithm: <a href="https://www.techopedia.com/definition/17137/genetic-algorithm">https://www.techopedia.com/definition/17137/genetic-algorithm</a>