Class Genetic

Abstract Public class **Genetic**

This class gives a foundation for creating genetic algorithm.

The documentation for the methods contained in this class includes briefs description of the *implementations*. Such descriptions should be regarded as *implementation notes*, rather than parts of the *specification*. Implementers should feel free to substitute other algorithms, so long as the specification itself is adhered to.

The genetic algorithm is inbuilt In the class and all the user has to do is to create a subclass of this class by defining the abstract function *calcFitness* .

Parameters for the constructer:

- **int** *num_genes_per_chromosome* :- This parameters specifies number of genes in the chromosome as defined by the user.
- **int** *num_chromosomes* :- This parameter specifies the size of the population in each generation as defined by the user.
- **double** *crossover_fraction* :- This parameter specifies the fraction of chromosomes to be cross-overed (used for reproduction) in a population.
- **double** *mutation_fraction* :- This parameter specifies the fraction of chromosomes to be mutated in a population.
- **Object** *Alleles_[]* :- This parameter specifies the array of alleles (the possible values for a gene) as specified by the user. (can be of any data type which is subclass of Object)
- **String** *type_of_genes* :- This parameter specifies whether the genes in a chromosome are supposed to be distinct("ordered") or non distinct ("not_ordered").

Attributes of the class:

- protected int numGenesPerChromosome;
- protected int numChromosomes;
- List<Chromosome> chromosomes – a list of elements of the datatype of Chromosome.
- protected double crossoverFraction;
- protected double mutationFraction;
- protected int[] rouletteWheel;
- protected int rouletteWheelSize;
- protected Object Alleles[];
- protected String typeofgenes – specifies whether "ordered" or "not_ordered".
- protected double Fitness[] – array containing the fitness of the whole population which is automatically updated.
- protected double highFitness – The highest fitness recorded until now.
- protected double lowFitness – The lowest fitness recorded until now.

- protected Chromosome HFChromosome – The chromosome with the highest fitness among the population is stored and updated automatically in this variable.
- protected Chromosome LFChromosome – The chromosome with the lowest fitness among the population is stored and updated automatically in this variable.

Method Summary

Public Object	**getGene(int chromosomeindex, int geneindex)** Returns the gene at the geneindex in the chromosome at the chromosomeindex of the list chromosomes.
Public void	**setGene(int chromosomeindex, int geneindex, Object Value)** Changes the value of the gene at geneindex of the chromosome at chromosomeindex to Object Value.
Public void	**evolve()** The main function which performs both crossover and mutation on the population. Contains functions for both "ordered" and "not_ordered" type of genes.
Abstract public double	**calcFitness(Chromosome chromosome)** The abstract function to be defined by the user to calculate the fitness of the chromosome.
Public void	**copy(Chromosome ch1, Chromosome ch2)** Rewrites ch2 as a copy of ch1.
Public boolean	**check(Object Gene, Object[] array)** Searches the specified array of Objects for the specified Object Gene.

Method Detail

getGene

```
public Object getGene(int chromosomeindex, int geneindex)
```

Returns the Object present at the index of value 'geneindex' in the chromosome whose index is 'chromosomeindex' in the list chromosomes.

Parameters:

```
chromosomeindex- index of the chromosome.
geneindex- index of the gene.
```

---

setGene

`public void` **`setGene`**`(int chromosomeindex, int geneindex, Object value)`

Rewrites the gene at the gene of index 'geneindex' of the chromosome of index 'chromosomeindex' in the list chromosomes to the Object value.

Parameters:

`chromosomeindex` – index of the chromosome.

`geneindex` - the index of gene.

`value` –the Object to be written.

---

evolve

`public void` **`evolve()`**

This function performs the crossover(both ordered and not_ordered) and the mutation(both ordered and not_ordered) on the population of the chromosomes and creates a new population of the next generation.

---

calcFitness

`abstract public double` **`calcFitness`**`(Chromosome chromosome)`

Calculates the fitness of the given chromosome according to the function defined by the user which is declared in the subclass. **This function should be written in such a way that the organism which is closer to the ideal organism has *higher fitness* than the one which is not.**

Parameters:

`chromosome` - the chromosome whose fitness is to be evaluated.

---

copy

`public void` **`copy`**`(Chromosome ch1, Chromosome ch2)`

Rewrites the chromosome ch2 as a copy of the chromosome ch1. (not by reference but by the value)

Parameters:

`ch1` – Chromosome to be copied.

`ch2` - Chromosome to be written.

---

check

`public boolean` **`check`**`(Object gene, Object[] array)`

Checks whether the Object gene is present in the Object[] array or not. If present it returns true else false.

Parameters:

`gene` – gene to be searched for.
`array[]` – array to be searched.
