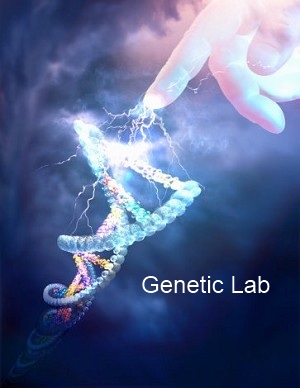
**Classes v0.2**



|  |  |
| --- | --- |
| Projecto: | **Life Inspiration** |
| Autor: | Genetic Lab |
| Data de preparação: | 2012-04-17 |
| Circulação: | Genetic Lab, Power Computing |

# Descrição sucinta das classe

**Classe Gene**

|  |
| --- |
| **Gene<T>** |
| - private T \_allele; |
| + public Gene (T valueAllele);  + public Gene(Gene<T> newGene);  + public T getAllele();  + public void setAllele(T allele);  + public String toString(); |

**Descrição:**

Classe que guarda a informação referente ao gene de um individuo. O valor será guardado dentro da variável \_allele, que pode ser acedida através do método getAllele(). Esta classe é genérica, o que significa que poderá guardar na variável \_allele qualquer tipo de informação.

**Construtor:**

O construtor public Gene (T valueAllele) servirá para criar um novo gene, sendo o valor a ser guardado no mesmo passado por parâmetro. O construtor public Gene(Gene<T> newGene) recebe um gene já definido e copia o mesmo para dentro da nova classe criada.

**Método:**

Com o método public String toString() podemos saber informações acerca do valor da variável \_allele, devolvendo assim uma string de zeros e uns referentes ao array de booleans guardado na mesma.

**Classe Chromosome**

|  |
| --- |
| **Chromosome**  *implements Iterable<Gene>* |
| - private ArrayList<Gene> \_genotype;  - private Individual \_individual; |
| + public Chromosome(Individual typeIndividual);  + public Chromosome(Chromosome newChromosome);  + private void \_inicializationGenotype();  + public Gene getGene(int index);  + public void setGene(Gene gene);  + public ArrayList<Gene> getGenotype();  + public void setGenotype(ArrayList<Gene> genotype);  + public Individual getIndividual();  + public void setIndividual(Individual individual);  + public String toString(); |

**Descrição:**

Classe que guardará as informações do cromossoma de um individuo. A variável \_genotype, que é um ArrayList de genes, guarda todos os genes definidos para este cromossoma, já que o mesmo poderá ser composto por vários genes. A variável \_individual guardará a informação referente ao tipo de individuo a que este cromossoma pertence.

**Construtor:**

Com o construtor public Chromosome(Individual typeIndividual) o cromossoma é criado com tantos genes como os definidos para o individuo que é passado por parâmetro, sendo que os genes do cromossoma são inicializados. Com o construtor public Chromosome(Chromosome newChromosome), o parâmetro de construção é um cromossoma, sendo que este é copiado para a nova instancia da classe.

**Métodos:**

O método public void setGene(Gene gene) permite adicionar um novo gene ao cromossoma e o método public Gene getGene(int index) devolve o gene definido no índex especificado no parâmetro.

O método public ArrayList<Gene> getGenotype() permite retirar o arraylist de genes que o cromossoma contem enquanto o método public void setGenotype(ArrayList<Gene> genotype) permite fazer a definição do mesmo arraylisdt, sendo passado por parametro um arraylist de genes para fazer a definição do cromossoma.

O método public Individual getIndividual() devolve o individuo ao qual o cromossoma pertence enquanto o método public void setIndividual(Individual individual) permite fazer a definição do individuo ao qual o cromossoma pertence.

Com o método public String toString() podemos retirar informação em forma de string com a informação do cromossoma, que será a informação de todos os genes que compõem o mesmo.

**Classe Individual**

|  |
| --- |
| *(abstract)* **Individual**  *implements Iterable<Chromosome>, Comparable* |
| * ArrayList <Chromosome> \_genome;   - Int \_sizeGenome;  - Int \_sizeGenotype;  - Int \_sizeAllelo;  - Int \_ageIndividual; |
| + public Individual();  + public Individual(int sizeGenome, int sizeGenotype, int sizeAllelo);  + public Individual(Individual newIndividual);  + public abstract int fitness();  + public abstract Boolean[] inicializationAllelo();  + public void inicializationGenome();  + public Chromosome getChromosome(int index);  + public void setChromosome(int index, Chromosome cromosome);  + public int getSizeGenotype();  + public void setSizeGenotype(int sizeGenotype);  + public int getSizeGenome();  + public void setSizeGenome(int sizeGenome);  + public int getSizeAllelo();  + public void setSizeAllelo(int sizeAllelo);  + public ArrayList<Chromosome> getGenome();  + public int getAgeIndividual();  + public void setAgeIndividual(int ageIndividual);  + public void incrementAge(); |

**Descrição:**

Classe que representa um individuo. A classe é abstracta para que possa ser utilizada em cada tipo de individuo definido na aplicação.

Recebe como parâmetro o tamanho do genoma, tamanho do genótipo e tamanho dos alelos.

É composta por um ArrayList com elementos do tipo Chromossome, chamados genome.

**Construtor:**

Com a utilização do construtor public individua() não são passados parâmetros para a construção do mesmo, sendo que as variáveis são inicializadas com valores por defeito que estão definidas na classe Population.

Com o construtor public Individual (int sizeGenome, int sizeGenotype, int sizeAllelo) são passados por parâmetro o tamanho do genoma, do genotype e do tamanho do alelo.

Com o construtor public Individual(Individual newIndividual) este recebe um individuo como parâmetro, sendo que o novo individuo criado através deste construtor irá receber os parâmetros do individuo passado por parâmetro, sendo que o genoma é também copiado para este novo individuo criado.

**Métodos:**

Os métodos, public abstract int fitness() e public abstract Boolean[] inicializationAllelo() são os métodos abstractos que terão que ser implementados em cada tipo especifico de individuo.

O método public void inicializationGenome() será o método que inicializa o genoma do indivíduo.

O método public Chromosome getChromosome(int index) permite retirar um cromossoma do genoma do individuo.

O método public void setChromosome(int index, Chromosome cromosome) permite definir o cromossoma especifico do genoma do individuo.

O método public int getSizeGenotype() devolve o tamanho do Genotype (Número de genes por cromossoma).

O método public void setSizeGenotype(int sizeGenotype) permite fazer a definição do número de genes por cromossoma.

O método public int getSizeGenome() devolve o tamanho do genoma do individuo (número de cromossomas).

O método public void setSizeGenome(int sizeGenome) permite fazer a definição do tamanho do genoma do individuo (número de cromossomas).

O método public int getSizeAllelo() devolve o tamanho do allelo do individuo.

O método public void setSizeAllelo(int sizeAllelo) pertmite fazer a definição do tamanho do allelo do individuo.

O método public ArrayList<Chromosome> getGenome() devolve o genoma do individuo, ou seja, um arraylist com todos os seus cromossomas.

O método public int getAgeIndividual() devolve a idade actual do individuo.

O método public void setAgeIndividual(int ageIndividual) permite definir a idade do individuo.

O método public void incrementAge() permite incrementar a idade do individuo.

**Classe Solver**

|  |
| --- |
| *(abstract)* **Solver** |
| - private Population \_parentsPopulation;  - private Population \_sonsPopulation;  - private int \_sizePopulation;  - private int \_sizeGenotype;  - private int \_sizeGenome;  - private int \_sizeAllelo;  - private Individual \_prototypeIndividual;  - private StopCriterion \_stopCriterion;  - private int \_numberIteractions;  - private ArrayList<Operator> \_operators;  - private EventsSolver \_eventSolver; |
| + public Solver(ArrayList<Operator> operators, EventsSolver eventSolver);  + public Solver(int sizePopulation, int sizeAllelo, Individual prototypeIndividual, StopCriterion stopCriterion, ArrayList<Operator> operators, EventsSolver eventSolver);  + public Solver(Population population, StopCriterion stopCriterion, ArrayList<Operator> operators, EventsSolver eventSolver);  + public void run() throws SolverException, SonsInicialitazionException; |

**Construtor:**

O construtor public Solver(ArrayList<Operator> operators, EventsSolver eventSolver) é o construtor do solver.

O construtor public Solver(int sizePopulation, int sizeAllelo, Individual prototypeIndividual, StopCriterion stopCriterion, ArrayList<Operator> operators, EventsSolver eventSolver) Construtor gera uma população de forma aleatória.

O construtor public Solver(Population population, StopCriterion stopCriterion, ArrayList<Operator> operators, EventsSolver eventSolver) passa uma população como parâmetro.

O método public void run() throws SolverException, SonsInicialitazionException faz correr o Solver, ou seja, aplica os operadores geneticos sobre a população ate encontrar o individuo com o fitness desejado ou atingir o numero máximo de gerações pré-determinado pelo utilizador.

**Classe Population**

|  |
| --- |
| *Population*  *implements Iterable<Individual>* |
| * ArrayList <Individual> \_population;   - Int \_sizePopulation;  - Int \_sizeGenotype;  - Int \_sizeGenome;  - Int \_sizeAllelo;  - Individual \_prototypeIndividual; |
| + Random RANDOM\_GENERATOR;  + public Population(Individual prototypeIndividual);  + public Population(Population newPopulation);  + public Population(int sizePopulation, int sizeGenome, int sizeGenotype, int sizeAllelo, Individual prototypeIndividual);  + public Population(int sizePopulation, int sizeGenome, int sizeGenotype, int sizeAllelo, Individual prototypeIndividual, boolean initializesPopulation)  + public Individual getIndividual(int index);  + public void setIndividual(int index, Individual individual);  + public int getSizePopulation();  + public int setSizePopulation();  + public int getTypePopulation();  + public void setTypePopulation(individual prototypeIndividual);  + public void addIndividual(Individual individual);  + public ArrayList<Individual> getPopulation();  + public void setPopulation(ArrayList<Individual> population);  + public int getSizeAllelo();  + public void setSizeAllelo(int sizeAllelo);  + public int getSizeGenotype();  + public void setSizeGenotype(int sizeGenotype);  + public int getSizeGenome();  + public void setSizeGenome(int sizeGenome); |

**Descrição:**

A classe Population que representa uma população de indivíduos.

Recebe como parâmetros o tamanho da população, o tamanho do genoma, tamanho do genótipo, tamanho dos alelos e o tipo da população(ex:boolean).

É composta por um ArrayList de elementos do tipo Individual.

A classe public static final Random RANDOM\_GENERATOR = new Random() é responsável pela geração de números aleatórios.

**Construtores:**

O construtor public Population(Individual prototypeIndividual) é o construtor por defeito, que cria uma população com os valores definidos também por defeito.

O constructor public Population(int sizePopulation, int sizeGenome, int sizeGenotype, int sizeAllelo, Individual prototypeIndividual) é responsável pela criação de uma população com todos os parâmetros e gera a população aleatoriamente.

O construtor public Population(int sizePopulation, int sizeGenome, int sizeGenotype, int sizeAllelo, Individual prototypeIndividual, boolean initializesPopulation) é idêntico ao anterior, mas com a opção de criar uma população aleatória ou não.

O método public Population(Population newPopulation) Inicialização de uma população de forma aleatória.

O método public Individual getIndividual(int index) devolve o indivíduo que está no índex.

O método public void setIndividual(int index, Individual individual) permite definir o indivíduo que está no índex.

O método public int getSizePopulation() devolve o tamanho da população.

O método public void setSizePopulation(int sizePopulation) perimite definer o tamanho da população.

O método public Individual getTypePopulation() devolve o tipo de indivíduos que estão na população.

O método public void setTypePopulation(Individual prototypeIndividual) permite definir o tipo de indivíduos que estão presentes na população.

O método public void addIndividual(Individual individual) permite adicionar um novo individuo à população.

O método public ArrayList<Individual> getPopulation() devolve o array list com os individuos.

O método public void setPopulation(ArrayList<Individual> population) permite definir o array list dos indivíduos na população.

O método public int getSizeAllelo() devolve o tamanho do alelo.

O método public void setSizeAllelo(int sizeAllelo) define o tamanho do alelo.

O método public int getSizeGenotype() devolve o tamanho do genotyoe.

O método public void setSizeGenotype(int sizeGenotype) define o tamanho do genotype.

O método public int getSizeGenome() devolve o tamanho do genoma.

O método public void setSizeGenome(int sizeGenome) define o tamanho do genoma.

**Classe Genetic**

|  |
| --- |
| *abstract Genetic e*xtends Operator |
| - static final public Random RANDOM\_GENERATOR; |
| + public abstract Population execute(Population population); |

Classe abstracta que contém um atributo e um método que é comum em todas as classes que herdam a classe abstracta.

**Classe Flipbit**

|  |
| --- |
| *Flipbit*  **Extends Mutation** |
| - static final int SIZE\_TOURNAMENT\_DEFAULT;  - static final boolean DEFAULT\_REMOVE\_INDIVIDUAL\_FROM\_POPULATION;  - final private int \_sizeTournament;  - private boolean \_removeIndividualFromPopulation; |
| + public Flipbit();  + public Flipbit(double probability);  + private Boolean[] \_doFlipBit(Boolean[] allelo); |

**Descrição da classe:**

Classe que recebe uma probabilidade como parâmetro, que define a probabilidade de um bit ser trocado. Serve para corrigir os bits dos indivíduos que não cabem na mochila de forma a ajustar o seu peso.

**Constructor:**

O construtor public Flipbit(double probability), recebe uma probabilidade que é a probabilidade dos bits serem alterados.

**Classe Mutation**

|  |
| --- |
| *Mutation*  **Extends Genetic** |
| - static final double PROBABILITY\_BY\_DEFAULT;  - protected double probability; |

Classe abstracta que contém dois atributos que são comuns em todas as classes que herdam a classe abstracta

**Classe Roulette**

|  |
| --- |
| *Roulette*  **Extends Selection** |
| - double ponteiro; |
| + public Roulette();  + public Roulette(int dimensionsNewPopulation); |

**Descrição da classe:**

A classe atribui uma percentagem a cada individuo de uma população com base no fitness e no total de fitness da população. Junta os indivíduos em linha de forma a criar percentagens acumuladas, com base na ordem dos indivíduos. Gera um número real aleatório entre 0 e 1, e um ponteiro que aponta para o individuo de acordo com o número gerado.

**Construtor:**

O Construtor Public Roulette (int dimensionsNewPopulation) com parâmetro, que recebe a dimensão da nova população.

**Classe SUS**

|  |
| --- |
| *SUS*  **Extends Selection** |
| - double offset;  - double ponteiro; |
| + public SUS();  + public SUS(int dimensionsNewPopulation);  + private double calculateOffset(Population population); |

**Descrição da classe:**

A classe SUS junta todos os indivíduos de uma população, somando o fitness do individuo anterior com o seu, numa linha de indivíduos.

Gera um número aleatório entre 0 e o número da soma do fitness dos indivíduos, para ser o ponto de partida.

Define o offset que acrescenta ao ponto de partida e selecciona o individuo.

**Construtor:**

O construtor public SUS(int dimensionsNewPopulation) com o parâmetro recebe a dimensão da nova população.

**Métodos:**

O método private double calculateOffset(Population population) calcula o offset, total fitness/total indivíduos da população.

**Classe Selection**

|  |
| --- |
| *abstract Selection extends Genetic* |
| - static final int DIMENDIONS\_NEW\_POPULATION\_DEFAULT;  - protected int \_dimensionsNewPopulation; |

Classe abstracta que contém dois atributos que são comuns em todas as classes que herdam a classe abstracta.

**Classe Tournament**

|  |
| --- |
| *Tournament*  **Extends Selection** |
| - static final int SIZE\_TOURNAMENT\_DEFAULT;  - static final boolean DEFAULT\_REMOVE\_INDIVIDUAL\_FROM\_POPULATION;  - final private int \_sizeTournament;  - private boolean \_removeIndividualFromPopulation; |
| + public Tournament();  + public Tournament(int dimensionsNewPopulation, int sizeTournament);  + public boolean isRemoveIndividualFromPopulation();  + public void setRemoveIndividualFromPopulation(boolean removeIndividualFromPopulation); |

**Descrição da classe:**

A classe torneio permite, seleccionar os melhores indivíduos de uma população utilizando o método de torneio.

Ou seja, os indivíduos são seleccionados de forma aleatória da população de pais, por defeito, o primeiro individuo é o best Individuo, e é removido da lista de candidatos. Quando comparados dois indivíduos da lista de candidatos a best (com base no seu fitness), se o individuo continuar a ser o best individuo, o candidato a best é removido para a população, caso contrario o que tiver melhor fitness passa a ser o best.

**Constructor:**

Este construtor **public Tournament(int dimensionsNewPopulation, int sizeTournament**), recebe a dimensão da nova população que vai ser criada e o número de indivíduos que entram no torneio.

**Métodos:**

**public boolean isRemoveIndividualFromPopulation();**

**public void setRemoveIndividualFromPopulation(boolean removeIndividualFromPopulation);**

Estes métodos servem para remover o individuo para a população.

**Classe StopCriteriation**

|  |
| --- |
| *StopCriteriation* |
| - private int \_numberIteractions;  - private Double \_goodFiteness; |
| + public class StopCriterion;  + public StopCriterion(int numberIteractions, Double goodFiteness);  + public int getNumberIteractions();  + public void setNumberIteractions(int numberIteractions);  + public Double getGoodFiteness();  + public void setGoodFiteness(Double goodFiteness) |

Descrição: Classe em que é definido o critério de paragem.

O construtor public StopCriterion(int numberIteractions, Double goodFiteness) é o constructor que recebe o numero de Iterações e o valor do melhor fitness.

O método public int getNumberIteractions() devolve o número de iterações.

O método public void setNumberIteractions(int numberIteractions) define o numero de iterações.

O método public Double getGoodFiteness() devolve o valor do melhor fitness de um individuo.

O método public void setGoodFiteness(Double goodFiteness) define o valor do melhor fitness de um indivíduo.

**Classe Statistics**

|  |
| --- |
| **Statistics** |
| - final private Population \_population;  - private Double[] \_arrayFitness;  - double media;  - double soma;  - double somatorio;  - double variancia; |
| + public Statistics(Population population);  + public Double getMediaFitnessPopulation();  + public Double getVarianciaPopulation();  + public Double getDesvioPadraoPopulation(); |

Descrição: Classe que servirá para fazer a análise estatistica dos valores da população que é recebida no parâmetro do construtor da classe.

O construtor public Statistics(Population population) recebe como parâmetro a população a ser analisada estatisticamente.

O método public Double getMediaFitnessPopulation() devolve a média de fitness da população, ou seja, devolve a soma de todos os fitness's dos individuos da população dividido pelo total de indivíduos da mesma.

O método public Double getVarianciaPopulation()devolve o valor da variância da população em análise.

O método public Double getDesvioPadraoPopulation() devolve o valor do desvio padrão da população.

**Classe PopulationUtils**

|  |
| --- |
| **PopulationUtils**  *implements Comparator<Individual>* |
| - final Population \_\_newPopulation;  - int \_\_bestFitness;  - int \_\_numberOfIndividualsWithBestFitness;  - final ArrayList<Individual> \_\_newArrayIndividual  - int \_\_countIndividual;  - int \_\_indexIndividual;  - int \_\_numberIndividualToReturn;  - final int NUMBER\_INDIVIDUALS\_TO\_GET\_FROM\_HALL\_OF\_FAME;  - final int FIRST\_INDIVIDUAL;  - int \_\_totalFitness;  - final Population \_\_newPopulation; |
| + public static Population getHallOfFame(Population population, int sizeHallOfFame);  + public static void orderPopulation(Population population);  + public static int getNumberIndividualsWithBestFitness(Population population);  + public static ArrayList<Individual> getArrayIndividualsRandom(Population population, int numberIndividual, boolean removeIndividualFromPopulation);  + public static int getBestFitness(Population population);  + public static int getFitnessTotal(Population population);  + public static Population getHallOfFameWithoutDuplicateIndividuals(Population population, int sizeHallOfFame); |

Descrição: Classe que servirá de apoio à classe Population para que da mesma possam ser retiradas informações acerca dos seus indivíduos, fitness's, hall of fame's e afins.

O método public static Population getHallOfFame(Population population, int sizeHallOfFame)devolve uma população que só irá conter os melhores indivíduos da população recebida por parâmetro.

O método public static void orderPopulation(Population population) ordena a população, ou seja, os seus individuos pelo seu fitness, do melhor para o pior.

O método public static int getNumberIndividualsWithBestFitness(Population population) devolve o número de individúos que tem o fitness máximo na população.

O método public static ArrayList<Individual> getArrayIndividualsRandom(Population population, int numberIndividual, boolean removeIndividualFromPopulation) devolve um ArrayList de individuos da população, sendo que esse individuos são uma amosta ao acaso da população em causa.

O método public static int getBestFitness(Population population) devolve o valor de fitness mais alto da população.

O método public static int getFitnessTotal(Population population) devolve o fitness total da população (soma do fitness de todos os individuos da população).

O método public static Population getHallOfFameWithoutDuplicateIndividuals(Population population, int sizeHallOfFame) devolve uma população que não irá conter individuos duplicados.

**Classe Ponteiro**

|  |
| --- |
| **Ponteiro** |
| - int totalFitness;  - double ponteiro; |
| + public static double pontoAleatorio(int totalFitnessPopulacao);  + public static Individual devolveIndividuoParaOndeOPonteiroAponta(double ponteiro, Population population) throws PonteiroForaDoLimiteException; |

Descrição: Classe Ponteiro que será um ponteiro para a população. Este ponteiro será calculado aleatoriamente e apontará para um individuo da população passada por parâmetro.

O método public static double pontoAleatorio(int totalFitnessPopulacao)devolve um ponto aleatório entre 0 e o total de fitness da população.

O método public static Individual devolveIndividuoParaOndeOPonteiroAponta(double ponteiro, Population population) throws PonteiroForaDoLimiteException devolve o individuo para onde o ponteiro aponta.

**Classe UniformCrossover**

|  |
| --- |
| **UniformCrossover**  *Extends Recombination* |
| - private Population sons;  - private Individual parent1, parent2, son1, son2;  - private boolean[] mask; |
| + private Individual aplicaMask(Individual parent, boolean[] mask)  + private void aplicaMask(Individual parent1, Individual parent2, boolean[] mask) |

Descrição: Classe que irá gerar uma cadeia de bits, ou mascara, para determinar quais os bits que vão ser trocados.

O método Individual aplicaMask, recebe um pai, aplica-lhe a máscara e devolve o filho com essa máscara

O método aplicaMask recebe dois pais, copia os seus alelos para os filhos e só fará a troca de genes na posição X dos filhos caso a máscara dessa posição seja true

**Classe Truncation**

|  |
| --- |
| **Truncation**  *Extends Replacement* |
| + public Truncation(int dimensionsNewPopulation);  + public Truncation(); |

Descrição: Classe que implementa o operador de substituição truncation. Este operador recebe duas populações, junta as duas numa só população, ordena a nova população por fitness de indivíduos do melhor para o pior e devolve uma nova população que será os melhores individuos das duas populações que foram juntas.

O construtor public Truncation(int dimensionsNewPopulation) passa por parâmetro a dimensão da nova população.

O construtor public Truncation() cria um novo Truncation com o número de individuos.

**Classe Tournament**

|  |
| --- |
| **Tournament**  *Extends Replacement* |
| - static final int SIZE\_TOURNAMENT\_DEFAULT;  - final private int \_sizeTournament; |
| + public Tournament();  + public Tournament(int dimensionsNewPopulation, int sizeTournament); |

Descrição: Operador que pega numa população e faz lutar os induvíduos para ver qual o melhor. No final coloca os melhores numa nova população.

O construtor public Tournament() é o construtor por defeito do tournament.

O construtor public Tournament(int dimensionsNewPopulation, int sizeTournament é o construtor do tournament que recebe como parâmetros dimensionsNewPopulation e sizeTournament.