**Functionality Outline**

**Mendelian Simulation**

**Programming Assignment 2**

**April 2, 2018**

**Prepared By**

**Atchima Klomkaew**

**ak0061@uah.edu**

**Prepared for**

**Dr. Rick Coleman**

**CS 307, Object Oriented Programming**

**Computer Science Department**

**University of Alabama in Huntsville**

1. **System Overview………………………………………….…………………….…2**
2. **Relevant Terms and Acronyms………………………….……….…………….…2**
3. **Object Functionality....…………………………………..………..……………….2**
   1. **Main**
      1. **Main()……………………..………….………………………....2**
   2. **MendelianSimulation**
      1. ***InitializeSimulation*...…………………………..………….…....2**
      2. ***RunSimulation*……………………..………..…………..……....2**
   3. **OrganismFactory**
      1. ***CreateOrganism.*………………………..………….…...…….....3**
      2. ***PrintParentData*……………………….…………..……....….....3**
   4. **Organism**
      1. ***LoadData………....*...…………………………..……....….........3**
      2. ***PrintData……........*...…………………………..……....….........3**
   5. **MasterGeneFactory**
      1. ***CreateMasterGene.*...……………………..……....…......……...3**
   6. **MasterGene**
      1. ***LoadData………....*...……………………..……....…......……...3**
      2. ***PrintData………....*...…………………..……....…......………...3**
   7. **ChromosomeFactory**
      1. ***CreateChromosome...*...……....……..……....…......…………....3**
   8. **Chromosome**
      1. ***CreateGene..………...*...……....……..……....…......…………...4**
      2. ***LoadData…………....*...……....……..……....…......…………...4**
   9. **Gene**
      1. ***LoadData…………....*...……....……..……....…......…………...4**
   10. **Offspring**
   11. **OffspringReader**
       1. ***PrintOffspring…………....*.......…..……....…......…..…..……...4**
   12. **Breed**
       1. ***GenerateOffspring…………....*....……..……....…....…..……...4**
       2. ***GetCorrectFormGene…………..*…..……....…..................*.*…...5**
   13. **BinaryTree**
       1. ***SearchTree…………....*...…….…..……....…....…..…………...5**
       2. ***Insert....*...…………………..……....…....…..……………..…...5**
       3. ***Insert....*...……………………..……....…..……………..….......5**
       4. ***PrintOne....*...……………....…....…..…………….…………....5**
       5. ***PrintTree....*...……………......…..……………..……..………...5**
       6. **PrintAll……………......…..……………..……..………….…...5**
4. **System Overview**

This project shall stimulate the processes of inheritance incorporating the Mendelian laws of genetics. The simulation shall read from the input file to identify the types of chromosomes, gene, genes’ definitions along with its crossover chance, and parents’ genotypes. It shall request the user to determine the total amount of offspring to generate. The parents’ genes will cross with one another in order to form an offspring, and the result will be similar to Mendel’s pea plants simulation (second generation). The phenotype ratio of the result is 3:1 for dominant gene and recessive gene respectively. The genotype ratio is 1:2:1 for homozygous dominant, heterozygous dominant, and homozygous recessive respectively. The output of this simulation shall include the amount of offsprings’ genotype generated for each gene, which should be in the ratio mentioned.

1. **Relevant Terms and Acronyms**

Mendelian laws of genetics include law of segregation, law of independent assortment, and law of dominance. The upper-case letter signifies dominant allele while lower-case letter signifies recessive allele. Genotype is the genetic makeup of an organism, i.e. TT Ww Ss cc. On the other hand, phenotype is the organism’s physical appearance, e.g. tall, wrinkled, green, and white. The crossover chance depends on the distance between the two genes in one strand of chromosome. The greater the distance, the greater the chance of the genes to get crossed over.

1. **Object Functionality**
   1. **Main**
      1. **Main()**

Instantiate MendelianSimulation object

Call MendelianSimulation::InitializeSimulation

Call MendelianSimulation::RunSimulation

* 1. **MendelianSimulation**
     1. ***InitializeSimulation()***

Query user for name of the data file

Create instance of GeneticsSimDataParser (*parser*)

Get data to define each object from data parser

Create instance of MasterGeneFactory

For each gene in GeneticsSimDataParser::getGeneCount

Call MasterGeneFactory::CreateMasterGenes

Parameter - parser

Define and generate parents

Create instance of OrganismFactory

For two parents

Call OrganismFactory::CreateOrganism()

* + 1. ***RunSimulation()***

Query user for amount of offspring and set it to *numOffsprings*

Create a new instance of Breed

Call Breed::GenerateOffspring

Parameters - numOffsprings, orgFact, binaryTree

Create a new instance of OffspringReader

Call OffspringReader::PrintOffspring

* 1. **OrganismFactory**
     1. ***CreateOrganism***

Get DataParser singleton instance

Get ChromosomeFactory singleton instance

Create a new instance of Organism

Call Organism::LoadData

For each chromosomes

Call ChromosomeFactory::CreateChromosome

Add the chromosome to the organism

Add the newly generated instance to the vector *orgCollection*

Return the Organism

* + 1. ***PrintParentData***

For all the elements in *orgCollection*

Call Organism::PrintData

* 1. **Organism**
     1. ***LoadData(parser : GeneticsSimDataParser, chromoCount : int)***

Call Organism::SetGenotype

Call Organism::SetGenusName

Call Organism::SetSpeciesName

Call Organism::SetCommonName

For all the function above, pass the corresponding function

from parser to obtain the data

* + 1. ***PrintData***

Print all the Organism data in a correct format

* 1. **MasterGeneFactory**
     1. ***CreateMasterGene***

Create a new instance of MasterGene

Call MasterGene::LoadData

Return the MasterGene

* 1. **MasterGene**
     1. ***LoadData(parser : GeneticsSimDataParser)***

Call all the set functions and correspond each to the function from

parser to obtain the data

* + 1. ***PrintData***

Print all the MasterGene data in a correct format

* 1. **ChromosomeFactory**
     1. ***CreateChromosome***

Create a new instance of Chromosome

Call Chromosome::LoadData

If building parent 1

Call the data parser and get the corresponding data

for parent 1 (GeneticsSimDataParser::getP1Chromosome)

Else if building parent 2

Call the data parser and get the corresponding data

for parent 2 (GeneticsSimDataParser::getP2Chromosome)

Get the MasterGene represented by each character in the strand

Call Chromosome::CreateGene

Add the returned gene to the chromsome’s vector of Gene

Add the newly created instance to the vector *chromoCollection*

Return the chromosome

* 1. **Chromosome**
     1. ***CreateGene***

Create a new instance of Gene

Call Gene::LoadData

Add the newly generated instance to the vector *geneCollection*

* + - 1. **LoadData*(parser : GeneticsSimDataParser)***

Call all the set function and correspond each to the function from

parser to obtain the data

* 1. **Gene**
     1. ***LoadData(parser : GeneticsSimDataParser)***

Call all the set function and correspond each to the function from

parser to obtain the data

* 1. **Offspring**

Inherits from Organism

* 1. **OffspringReader**
     1. ***PrintOffspring(tree : BinaryTree)***

Put the output in the right format

For each gene

Initialize counters for homozygous dominant, heterozygous

dominant, and homozygous recessive offspring

For each offspring in the BinaryTree *tree*

If crossover flag is true, increment crossoverCount

Extract offspring genotype string

For each pair of gene letters in the string

Get two letters and check letters for:

(1) Upper-Upper, (2) Upper-Lower, and

(3) Lower-Lower

Increment the appropriate counter

Print resulting counters for this gene

Print total number of offspring with crossover

* 1. **Breed**
     1. ***GenerateOffspring(tree : BinaryTree, par1 : Organism, par2 : Organism, numOffspring : int, numChromo : int)***

For each offspring

For numChromo

Call Organism::GetRandChromoStrand from each

parent

Get cross over flag in each parent

For each letter in strand

Get a letter from each strand to create a gene

Call Breed::GetCorrectFormGene

Add this gene pair to offspring genotype str

Create an instance of Offspring

Set its genotype string to the one built

If crossover flag of either parent is true

Set offspring crossover flag to true

Else set it to false

Add offspring to collection of offspring in BinaryTree *tree*

* + - 1. ***GetCorrectFormGene***

Ensure uppercase letter/dominant allele is precedent of lowercase

letter/recessive allele

* 1. **BinaryTree**
     1. **Function *SearchTree(key : char, T : Node)***

If the node T is not null

If the key at node T is the same as the key, then return T

Else if the target key is less than the T’s key

Return BinaryTree::SearchTree(key, T->left)

Else

Return BinaryTree::SearchTree(key, T->right)

Else return NULL

* + - 1. ***Insert(key : char)***

Create a new instance of Node *newNode*

Initialize the node with the key value given

Set the left and right pointers to NULL

Return BinaryTree::Insert(newNode)

* + - 1. ***Insert(newNode : Node)***

If BinaryTree::SearchTree(newNode->key, root) is not NULL

Then increment the counter of the node and return to the

caller

Else

Find where to insert the node by comparing key values

Insert the node appropriately with cases of: (1) Inserting at

root, (2) Inserting on the left, and (3) Inserting on the right

* + - 1. ***PrintOne(T : Node)***

Print out T’s key and T’s counter in an appropriate format

* + - 1. ***PrintTree***

Call BinaryTree::PrintAll(root)

* + - 1. ***PrintAll(T : Node)***

If T is not NULL

// in-order traversal

PrintAll(T->left)

PrintOne(T)

PrintAll(T->right)

**There will be get and set functions defined for each private member variable.**