Functionality Outline

Simulation of Mendelian Laws of Genetics
Programming Assignment 2

Date: March 30th, 2018

Prepared by
Sean Mitchell
sm0077@uah.edu

Prepared for
Dr. Rick Coleman
CS 307 - 01: Object Oriented Programming in C++
Computer Science Department
University of Alabama in Huntsville

Contents

1 System Overview

1.1 UML Diagram

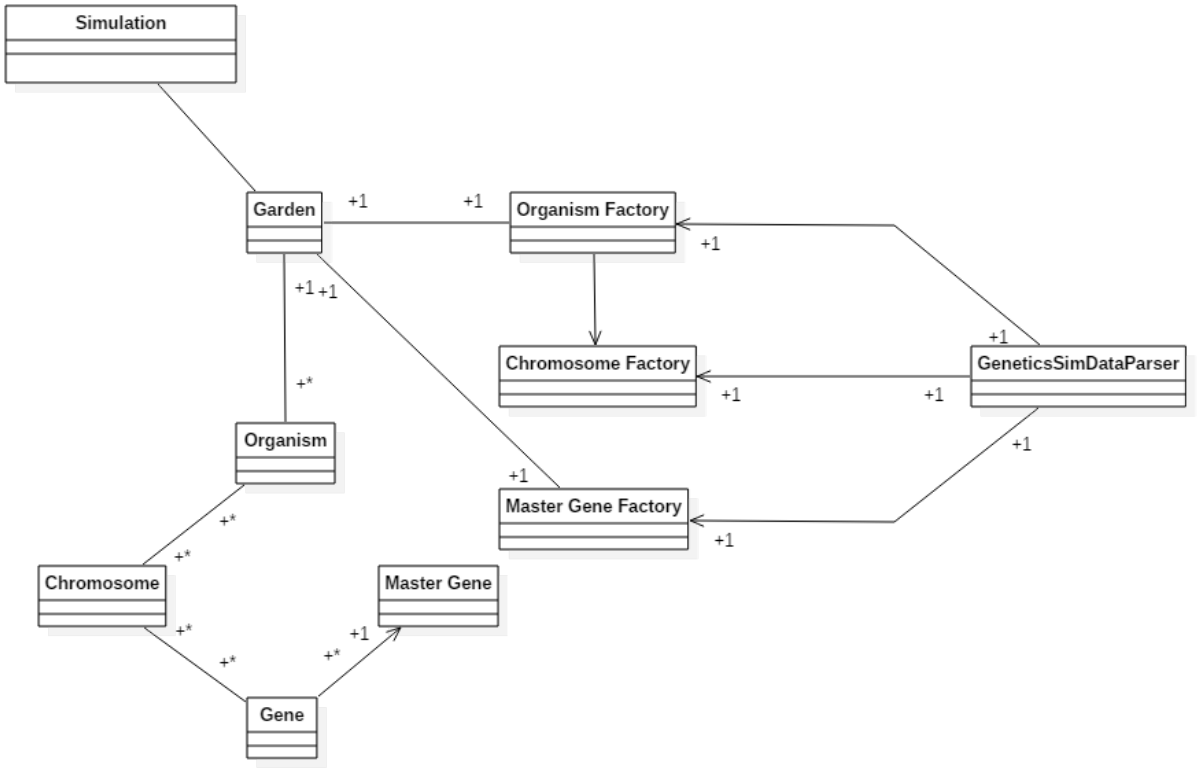

Figure 1: Preliminary UML Class Diagram

2 Class Descriptions

Simulation The Simulation class will perform three main tasks. The first task will be to ask the user how many offspring to create. The second task will be asking the user the name of the data file to read as input. Using the input from the first two tasks, the Simulation object will create an instance of Garden, passing the number of children to create and the input file as arguments.

Garden The Garden class will perform several tasks. The first task will use the data file passed as input to create two parent Organism objects. Once these two parents have been created, the Garden object will allocate data structures to store the children. Next, the Garden object will breed the two parents until the number of offspring created matches the input received from the Simulation object. Once all the offspring are created, the Garden object will sort the data structures. Finally the Garden object will return a report of the results of the simulation.

Parser The Parser object will read the attributes of each gene. This object will be called by the factories to return the data needed to for the factories to create their instances.

Organism The Organism class will be an instance of the object returned from the Organism Factory. The class will contain a Chromosome vector which will store all the Chromosome objects that the Organism has. Other member variables that class will have include both the common name and scientific name. The class will also have accessor and mutator functions to these member variables.

Organism Factory The Organism Factory will perform the creation of instances of Organism. The Organism Factory class is a simple factory. The Organism Factory shall use the Chromosome Factory to create chromosomes for organisms. The Organism Factory shall contain a function which returns an instance of an organism with fully defined genotype. This factory will also be a Singleton.

Chromosome The Chromosome class will be an instance of the object returned from the Chromosome Factory. Instances of Chromosome will store the genes for that particular chromosome.

Chromosome Factory The Chromosome Factory class will perform the creation of instances of chromosomes. This class will be a simple factory. The Chromosome Factory shall use the Master Gene Factory to create genes for chromosomes. This factory will also be a Singleton.

Gene The Gene class will be an instance of the object returned from the Gene Factory. The Gene class will hold a reference to the master gene and variables to hold the specific allele characters for that instance of the gene.

Master Gene The Master Gene class is a single instance for each type of gene. This instance will hold most of the information that is common to many genes.

Master Gene Factory The Master Gene Factory class will perform the creation of instances of both Master Gene and Gene. This class is a simple factory. This factory will also be a Singleton.

3 Background Infomation

In 1866 an Austrian monk named Gregor Mendel published a paper in Proceedings of the Natural Society of Brunn, in which he described the laws of genetics which he had formulated from his work cultivating, cross-breeding, and testing thousands of pea plants (Pisum sativum). Some of the traits he studied were plant stature (tall vs. dwarf), seed color (green vs. yellow), and seed pod texture (smooth vs. wrinkled). While getting little attention from the scientific community at first, his work later became the foundation for the entire science of genetics and today Mendel is known as the ”father of modern genetics.”

Mendel showed that for any particular trait, such as plant stature, there were actually two genes forming a matched gene pair to determine the trait. Each of the genes in the pair could be either of two types. One gene type produced tall plants. This he represented with a T. The other produced dwarf plants and he represented this with a t. Thus, to represent the genotype or genetic makeup of a plant with regard to stature he could use, TT, Tt, or tt. He found that if a plant had at least one ”tall” gene (represented as TT or Tt) it was always tall. This gene type he called dominant. Only when the genotype for a plants stature was represented as tt was it a dwarf plant. This gene type he called recessive.

Mendel came to this conclusion when he found that if he crossed a plant that always produced tall offspring, i.e. was pure-bred tall, with a plant that always produced dwarf offspring, i.e. was pure-bred dwarf, he got not medium sized plants but all tall plants. And, when he crossed these plants he got both tall and dwarf plants in a ratio of three tall to one dwarf. Mendel said that organisms have genes for traits in pairs. That when reproducing each parent plant contributes one of their pair of genes for each trait to each of the offspring. Which gene of the pair is contributed is purely random. This is easy to see if we look at a diagram representing the possible combinations. On the left we see the possible combinations when crossing a pure-bred tall (also called homozygous tall) with a pure-bred dwarf (also called homozygous dwarf) plant. All the offspring will be hybrid tall (also called heterozygous tall). When two of these hybrid plants are crossed we see that the phenotypes (visible traits) give a ratio of tall to dwarf plants of 3:1. The possible genotype combinations give a ratio of one homozygous tall to two heterozygous tall to one homozygous dwarf.

4 Object Functionality

4.1 CS_307_Main.cpp

4.1.1 Main()

Instantiate Simulation object
Call Simulation→Sim_Main()

4.2 Simulation.cpp

4.2.1 Simulation()

Declares member variables

4.2.2 ~Simulation()

Calls delete on all objects within the object

4.2.3 Sim_Main()

Ask for number of children as input
Store received input in int variable
Ask for name of input data file
Store received input in string variable
Instantiate Garden object
Call Garden → Garden_Main()
Call Sim_Report()

4.2.4 Sim_Report()

Queries the Garden object for the number of entries in its internal data structures
Prints out the results of this query to the terminal

4.3 Class Garden

4.3.1 Garden()

Declares member variables
Declares parent vector
Declares child vector
Declares report vector
Declares detailed report vector
Instantiate Organism Factory object
Instantiate Chromosome Factory object
Instantiate Master Gene Factory object
Instantiate master objects
Instantiate parent Organism objects
Store parent Organism objects in parent vector

4.3.2 ~Garden()

Calls delete on all objects within the object

4.3.3 Garden_Main()

Instantiate parent_1 object
store parent_1
Instantiate parent_2 object
store parent_2
Call Breed()

4.3.4 Breed()

Loop from 0 to number_of_children
    pass parent objects to organism factory
    Add that child from the organism factory to the children vector

4.3.5 Get_Child_Vector()

Returns the child vector

4.4 Class Parser

4.4.1 GeneticsSimDataParser()

Declares member variables
Instantiate parser objects

4.4.2 ~GeneticsSimDataParser()

Deletes parser objects

4.4.3 getGeneCount()

Returns m_iNumGenes

4.4.4 getGenus()

Returns m_sGenus

4.4.5 getSpecies()

Returns m_sSpecies

4.4.6 getScientificName()

Returns m_sSciName

4.4.7 getCommonName()

Returns m_sCommonName

4.4.8 getGeneData()

Open input file
Walk through file
    Set next gene to member variables
Close file

4.4.9 getParentGenotype()

Open input file
Walk through file
    Get parent genotype   Close file

4.5 Class Organism Factory

4.5.1 Organism_Factory()

Private constructor

4.5.2 Organism_Factory Instance()

Returns the instance of the organism factory

4.5.3 Breed(parent_1, parent_2)

Calls the chromosome factory object, passing the two parents as arguments Returns the new child organism

4.6 Class Organism

4.6.1 Organism()

Creates parent Organism object
Calls chromosome factory, passing the chromosomes the child organism received as arguments
Store chromosomes in the Organism's chromosome vector

4.6.2 ~Organism()

Calls delete on all objects within the object

4.6.3 Set_Name()

Sets the name field to the input of this function

4.6.4 Get_Name()

Returns the name field

4.6.5 Set_Attribute()

Sets the attribute field to the input of this function

4.6.6 Get_Attribute()

Returns the attribute field

4.6.7 Set_Gene()

Appends the input onto the gene vector

4.6.8 Get_Gene()

Returns the gene vector

4.7 Class Chromosome Factory

4.7.1 Chromosome_Factory()

Private constructor

4.7.2 Chromosome_Factory Instance()

Returns the instance of the chromosome factory

4.7.3 Breed(parent_1_chromosome, parent_2_chromosome)

Calls the master gene factory object, passing the two chromosomes as arguments
Returns the new child chromosome vector

4.8 Class Chromosome

4.8.1 Chromosome()

create chromosome vector

4.8.2 Get_Chromosome()

Returns the chromosome vector

4.9 Class Gene

4.9.1 Gene(allele_characters)

Stores allele characters in a string

4.9.2 ~Gene()

Calls delete on all objects within the object

4.9.3 Get_Allele_Characters

Returns the allele characters for that gene

4.9.4 Get_Name()

Returns the name field from the master gene pointer

4.9.5 Get_Attribute()

Returns the attribute field from the master gene pointer

4.9.6 Get_Gene()

Returns the gene field from the master gene pointer

4.10 Class Master Gene

4.10.1 Master_Gene()

Private constructor

4.10.2 Get_Name()

Returns the name field

4.10.3 Get_Attribute()

Returns the attribute field

4.10.4 Get_Gene()

Returns the gene field

4.11 Class Master Gene Factory

4.11.1 Master_Gene_Factory()

Private constructor

4.11.2 Master_Gene_Factory Instance()

Returns the instance of the chromosome factory

4.11.3 Create_Gene()

Takes two sets of chromosomes as input
Computes the cross of the two sets and if there is a crossover
assigns the correct master gene pointer to the new gene object
returns the gene