**Class Outline**

Simulation of Mendelian Laws of Genetics
**Programming Assignment** 2

Date: March 18th, 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

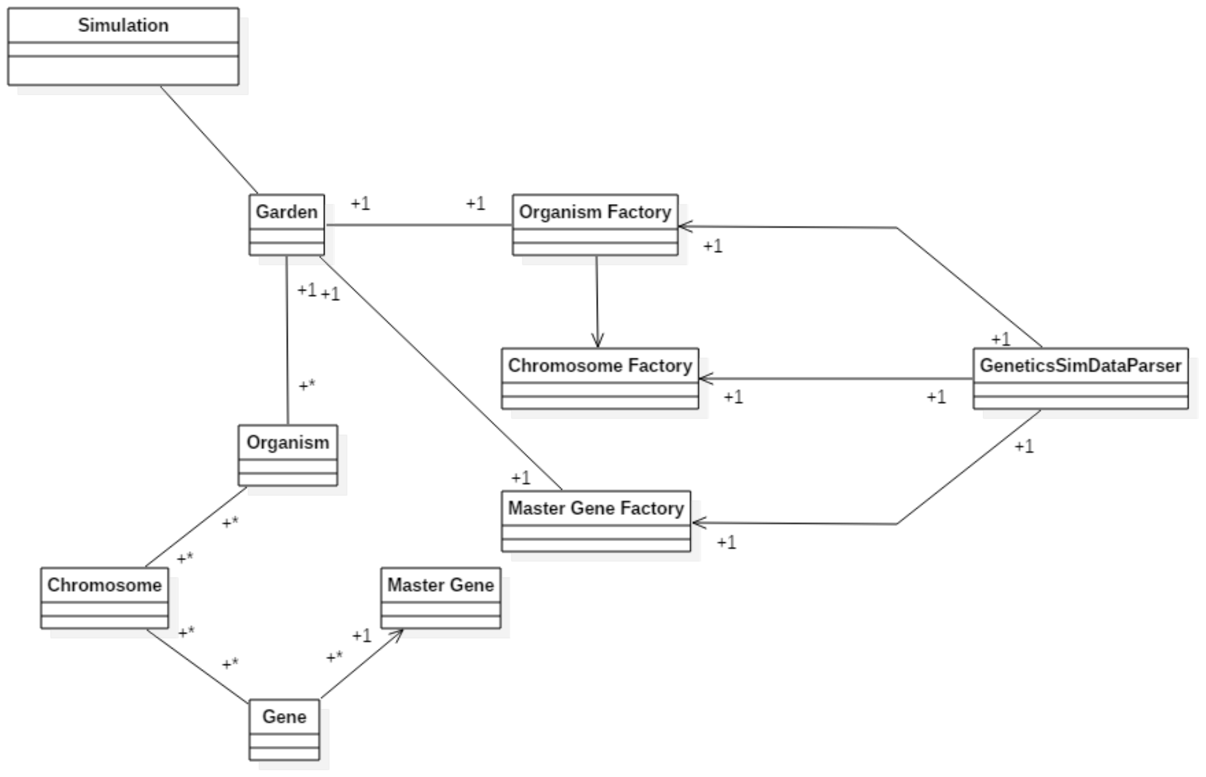

Figure 1: Preliminary UML Class Diagram

## 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 Information

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 Description

## 4.1 Class Simulation

### 4.1.1 Member Variables

**int number_of_children** Number of children to create

**string input_file** String the contains the name of the input data file

### 4.1.2 Member Functions

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

#### 4.1.2.1 Function Simulation()

Actions Performed- Constructor for the Simulation object

Arguments-None

Return Value- This function an instance of the Simulation class

#### 4.1.2.2 Function Sim_Start()

Actions Performed- This function starts the simulation, and queries the user for all the required inputs (data file, number of children)

Arguments-None

Return Value- This function returns void

#### 4.1.2.3 Function Sim_Report()

Actions Performed- This function returns a report of the simulation

Arguments-None

Return Value- This function returns void

## 4.2 Class Garden

### 4.2.1 Member Variables

**vector parents** Data structure to hold the parents

**vector master_genes** Data structure to hold the master genes

**vector children** Data structure to hold the children

**string input_file** String the contains the name of the input data file

### 4.2.2 Member Functions

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

#### 4.2.2.1 Function Garden()

Actions Performed-Default constructor

Arguments-string file_path

Return Value-void

#### 4.2.2.2 Function Create_Parents()

Actions Performed- This function calls the Organism Factory and stores the returned objects in the parent vector.

Arguments- This function does not take any arguments.

Return Value- This function returns void.

#### 4.2.2.3 Function Breed()

Actions Performed- This function calls the Organism Factory and stores the returned objects in the parent child

Arguments- This function does not take any arguments.

Return Value- This function returns void.

#### 4.2.2.4 Function Sort_Vector()

Actions Performed- This function walks through the children vector and finds the number of children that have specific gene combinations. When this is complete, it returns an array of ints, where each element is the number of children with that combination of genes

Arguments- This function does not take any arguments.

Return Value- This function returns an array of ints.

## 4.3 Class Parser

### 4.3.1 Member Variables

**ifstream inFile** Genetics sim data definition file

**int m_iNumGenes** Number of genes in the data file

**int m_iNextGeneNumber** Counter to next gene to read

**int m_iNextParentNumber** Counter to next parent genotype to read

**char m_sFileName** Data file name

**char m_sGenus** Genus name of this test organism

**char m_sSpecies** Species name of this test organism

**char m_sSciName** Full scientific name of the test organism

**char m_sCommonName** Common use name of the test organism

### 4.3.2 Member Functions

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

#### 4.3.2.1 Function GeneticsSimDataParser

Actions Performed-Default constructor

Arguments- This function takes a char holding the input file name as an argument

Return Value-This function returns an instance of the class

## 4.4 Class Organism

### 4.4.1 Member Variables

**vector genotype** vector that contains the chromosomes of the organism

**string common_name** Private member string holding the common name of the organism

**string scientific_name** Private member string holding the scientific name of the organism

### 4.4.2 Member Functions

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

**4.4.2.1 Function Organism()** Actions Performed-Constructor that handles the creation of the parent objects. Once the a parent is created, it gets added to the parent vector

Arguments-This function takes the return of the Parser object and creates a parent based on the contents of the data file

Return Value- This function returns an instance of the class

**4.4.2.2 Function Organism()** Actions Performed-Child constructor

Arguments-This function takes two parents as arguments and creates a child based on those attributes

Return Value- This function returns an instance of the class

## 4.5 Class Organism Factory

### 4.5.1 Member Variables

**Organism Factory * instance** instance of the factory

**string input_file** String the contains the name of the input data file

### 4.5.2 Member Functions

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

#### 4.5.2.1 Function Organism Factory() 
Actions Performed-Constructor

Arguments- This function takes the name of the input data file as an argument

Return Value- This function returns void.

#### 4.5.2.2 Function static *Get_Instance() 
Actions Performed-Function to get a pointer to the instance of this factory

Arguments- This function does not take any arguments.

Return Value- This function returns a pointer to the instance of this factory

#### 4.5.2.3 Function Build_Organism() 
Actions Performed-Builds an organism

Arguments- This function does not take any arguments.

Return Value- This function returns an organism object

## 4.6 Class Chromosome

### 4.6.1 Member Variables

**vector gene** vector that contains the genes of the organism

### 4.6.2 Member Functions

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

## 4.7 Class Chromosome Factory

### 4.7.1 Member Variables

**Chromosome Factory * instance** instance of the factory

### 4.7.2 Member Functions

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

#### 4.7.2.1 Function Chromosome Factory()

Actions Performed-Constructor

Arguments- This function takes the name of the input data file as an argument

Return Value- This function returns void.

#### 4.7.2.2 Function static *Get_Instance()

Actions Performed-Function to get a pointer to the instance of this factory

Arguments- This function does not take any arguments.

Return Value- This function returns a pointer to the instance of this factory

#### 4.7.2.3 Function Build_Chromosome()

Actions Performed-Builds a chromosome

Arguments- This function does not take any arguments.

Return Value- This function returns an chromosome object

## 4.8 Class Gene

### 4.8.1 Member Variables

**string gene_id** Private member string holding the specific allele characters for that instance of the gene.

**Master Gene Pointer** Private pointer to the master gene for that specific instance of the gene

### 4.8.2 Member Functions

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

**4.8.2.1 Function Gene()** Actions Performed- Default constructor. This constructor is used to create the objects for the master genes. The master genes will contain all the possible info about each genotype.

Arguments-None

Return Value- This function returns an instance of the class

**4.8.2.2 Function Gene()** Actions Performed- Overloaded constructor. This constructor creates a new gene based on the genes it receives from two parent Organisms.

Arguments- This function receives input of two alleles, one from each parent.

Return Value- This function returns an instance of the class

## 4.9 Class Master Gene

### 4.9.1 Member Variables

**string gene_trait** Private member string holding a string of the trait of that gene

**string gene_dom_symbol** Private member string holding the symbol of the dominant gene

**string gene_rec_symbol** Private member string holding the symbol of the recessive gene

**string gene_crossover** Private member string holding the percentage crossover chance of that gene

### 4.9.2 Member Functions

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

## 4.10 Class Master Gene Factory

### 4.10.1 Member Variables

**Gene Factory * instance** instance of the factory

### 4.10.2 Member Functions

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

#### 4.10.2.1 Function Gene Factory()

Actions Performed-Constructor

Arguments- This function takes the name of the input data file as an argument

Return Value- This function returns void.

#### 4.10.2.2 Function static *Get_Instance()

Actions Performed-Function to get a pointer to the instance of this factory

Arguments- This function does not take any arguments.

Return Value- This function returns a pointer to the instance of this factory

#### 4.10.2.3 Function Build_Gene()

Actions Performed-Builds a gene

Arguments- This function does not take any arguments.

Return Value- This function returns an gene object