Class Outline

**GeneticsSim**

Programming Assignment 2

March 12, 2016

**Prepared by**

Marcus Posey

[mlp0027@uah.edu](mailto:mlp0027@uah.edu)

**Prepared for**

Dr. Rick Coleman

CS 307, Object Oriented Programming

Computer Science Department

University of Alabama in Huntsville

Table of Contents

1. System Overview ……………………….. 1
2. Relevant Terms and Acronyms …………. 1
3. Object Description ……………………… 1
   1. Class Simulation …………………… 1
   2. Class Gene …………………………. 2
   3. Class GeneFactory ………………….. 3
   4. Class Organism …………………….. 3
   5. Class OrganismFactory …………….. 4
   6. Class Observable ….………………... 4
   7. Class IObserver ……………………. 5
   8. Class StatCounter …………………... 5
   9. Class MasterGene ………………….. 6
   10. Class MasterGeneFactory ……...… 7
   11. Class Chromosome …………...….. 7
   12. Class ChromosomeIterator ………. 7
   13. Class ChromosomeFactory ……..... 8
4. **System Overview**

The main objective of the project is to produce a program capable of simulating Mendelian laws of inheritance. These laws of inheritance govern what composes the genetic makeup of the offspring of two organisms. This task will be achieved by creating software models of basic components of a living organism. These include, but may not be limited to, alleles, genes, chromosomes, and the organism itself. By crossing the genotypes of two parent organisms that are to be read from an initial xml file, new genotypes that belong to offspring will be created. This data will then be recorded to acquire statistical representations of total gene and gene crossover counts among all children. All genetic data (i.e. that of parent organisms and offspring) will be presented on the screen in some manner—initially through a command-line interface and, in the future, possibly through a GUI.

1. **Relevant Terms and Acronyms**

* **Allele** – These are represented in the model by a single character that is either uppercase or lowercase. Uppercase letters dominant traits, while lowercase represent recessive traits. Pairs of alleles make up genes.
* **Gene –** Genes are pairs of alleles that influence the phenotype (visible characteristics) of an organism.
* **Chromosome** - The collection of genes that genetically describe an organism
* **Crossing Over** **–** The act of a segment of genes swapping places with the opposing segment of a chromosome
* **unordered\_map –** This is the STL implementation of a dictionary using a hash table. It assigns certain values to unique keys.
* **unique\_ptr** – This is the STL implementation of a managed pointer. It implements RAII to acquire possession of memory on the heap and free said memory when the container goes out of scope.

1. **Object Description**
   1. Class *Simulation*
      1. Member Variables

* **\_statCounter –** a StatCounter object. It observes Simulation and records/prints stats of produced offspring
* **\_offspringCount –** the number of offspring that will be created during the simulation
* **\_genes –** A collection of all MasterGene objects to be used by Gene objects
  + 1. Member Functions
       1. Function *getInstance*

**Actions Performed –** Constructs and/or returns a static Simulation object

**Arguments –** None

**Return Value –** a pointer to a static Simulation object

* + - 1. Function *run*

**Actions Performed –** Performs the simulation by generating the specified number of offspring

**Arguments –** None

**Return Value –** void

* + - 1. Function *init*

**Actions Performed –** Queries the user for the data file location and generation count and stores this data for Simulation::run to use

**Arguments –** None

**Return Value –** void

* 1. Class *Gene*
     1. Member Variables
* **\_allele1 –** One allele symbol (char) of a gene pair
* **\_allele2 –** One allele symbol (char) of a gene pair
* **­\_master –** A shared\_ptr to the MasterGene object which contains the gene’s shared details
  + 1. Member Functions
       1. Function *Gene*

**Actions Performed –** constructs a Gene object with an initial allele pair and reference to a MasterGene

**Arguments –**

**char a1 –** an allele symbol

**char a2 –** an allele symbol

**shared\_ptr<MasterGene> master –** the MasterGene containing shared genetic details

**Return Value -** None

* + - 1. Function *getPhenotype*

**Actions Performed –** determines the visible allele trait

**Arguments –** None

**Return Value –** phenotype as a string

* + - 1. Function *getZygosity*

**Actions Performed –** determines the zygosity of the two alleles

**Arguments –** None

**Return Value –** the zygosity of the gene as a string

* + - 1. Function *getAllelesString*

**Actions Performed –** concatenates the two allele symbols

**Arguments –** None

**Return Value –** returns a string containing the concatenated allele symbols

* + - 1. Function *toString*

**Actions Performed –** concatenates the zygosity, phenotype, and alleles into one string

**Arguments –** None

**Return Value –** returns a string containing the zygosity, phenotype, and alleles of the gene

**There are also get functions for the member variables.**

* 1. Class GeneFactory
     1. Member Variables

None

* + 1. Member Functions
       1. Function *getInstance*

**Actions Performed/Return Value** – returns a pointer to a static instance of the GeneFactory class. This is the only way to access GeneFactory, as it implements the singleton pattern.

**Arguments** – None

* + - 1. Function *createGene*

**Actions Performed** – Creates a Gene object using the supplied arguments

**Arguments** –

**char a1** – one allele of the Gene pair

**char a2** – one allele of the Gene pair

**Return Value** – The created Gene object

* 1. Class *Organism*
     1. Member Variables
* **\_genus –** the genus of the organism
* **\_species –** the species of the organism
* **\_name –** the common name of the organism
* **\_chromosomes –** a vector of Chromosome objects that compose the organism
* **\_serveCounter –** integer that indicates the index of the next chromosome to retrieve through ::serveChromosome
  + 1. Member Functions
       1. Function *Organism*

**Actions Performed –** constructs an organism object with an initial genus, species, and name but no chromosomes

**Arguments –**

**genus –** the genus of the organism

**species –** the species of the organism

**name –** the common name of the organism

**Return Value –** None

* + - 1. Function *addChromosome*

**Actions Performed –** adds a Chromosome object to the \_chromosomes vector

**Arguments –**

**Chromosome chromosome –** the chromosome to add to the organism’s chromosome vector

**Return Value –** void

* + - 1. Function *serveChromosome*

**Actions Performed –** retrieves the next Chromosome in \_chromosomes and increments \_serveCounter

**Arguments –** None

**Return Value –** returns the next Chromosome in \_chromosomes

* + - 1. Function *getChromosomeCount*

**Actions Performed/Return Value –** returns the number of chromosomes in the chromosome vector

**Arguments –** None

* + - 1. Function printDescription

**Actions Performed –** prints the genetic makeup of the organism to the command line

**Arguments –** None

**Return Value -** void

**There are also get functions for the member variables.**

* 1. Class OrganismFactory
     1. Member Variables

None

* + 1. Member Functions
       1. Function *getInstance*

**Actions Performed/Return Value -** returns a pointer to a static instance of the OrganismFactory class. This is the only way to access this class, as it implements the singleton pattern

* + - 1. Function *createOrganism*

**Actions Performed** – creates an Organism object with the provided arguments and then populates its chromosomes using ChromosomeFactory

**Arguments** –

**string genus** – the genus of the organism

**string species** – the species of the organism

**string name** – the common name of the organism

**Return Value** – returns an Organism object with complete genetic data

* 1. Class *Observable*
     1. Member Variables
* **\_observers –** a vector of IObservable objects
  + 1. Member Functions
       1. Function *addObserver*

**Actions Performed –**adds an observer to \_observers

**Arguments –**

**IObserver o –** an observer to add to the stored vector

**Return Value –** void

* + - 1. Function *removeObserver*

**Actions Performed –** removes an observer from \_observers

**Arguments –**

**IObserver o –** the observer to remove from the stored vector

**Return Value –** void

* + - 1. Function *notifyAll*

**Actions Performed –** sends the specified message to all observing objects

**Arguments –**

**T arg –** a message to send to all observers

**Return Value –** void

* + - 1. Function contains

**Actions Performed –** determines the existence of a stored IObserver object

**Arguments –**

**IObserver<T> o –** This is the observer to look for

**Return Value –** returns true if the Observable object is being observed by the supplied argument; false otherwise

* + - 1. Function getObserverCount

**Actions Performed/Return Value –** returns the number of IObservers that are observing the object

**Arguments –** None

* 1. Class *IObserver*
     1. Member Variables

**None**

* + 1. Member Functions
       1. Function pure virtual *notify*

**Actions Performed –** processes some message

**Arguments –**

**T arg –** some message to process

**Return Value –** void

* 1. Class *StatCounter*
     1. Member Variables
* **\_crossoverCount –** the number of times a crossover occurred during the simulation
* **\_geneCounts –** an unordered\_map<string, unordered\_map<string, int>> that maps a gene description to a map of a gene string representation and its associated count
  + 1. Member Functions
       1. Function *notify*

**Actions Performed –** stores the gene of the received organism and updates the crossover counter if necessary

**Arguments –**

**Organism arg –** an organism with a genotype

**Return Value –** void

* + - 1. Function *printStats*

**Actions Performed –** prints the crossover and gene counts to the command window

**Arguments –** None

**Return Value –** void

* 1. Class MasterGene
     1. Member Variables
* **\_trait** – the trait that the gene encodes for
* **\_dominantAllele** – the name of the dominant allele of the pair
* **\_recessiveAllele** – the name of the recessive allele of the pair
* **\_dominantSymbol** – the symbol (char) of the dominant allele of the pair
* **\_recessiveSymbol** – the symbol (char) of the recessive allele of the pair
* **\_crossoverChance** - the percentage chance that the gene will “cross over”
  + 1. Member Functions
       1. Function *MasterGene*

**Actions Performed –** initializes the member variables to the values defined by the supplied arguments

**Arguments –**

**string trait** – the trait that the gene encodes for

**string dominantAllele** – the name of the dominant allele of the pair

**string recessiveAllele** – the name of the recessive allele of the pair

**char dominantSymbol** – the symbol (char) of the dominant allele of the pair

**char recessiveSymbol** – the symbol (char) of the recessive allele of the pair

**char crossoverChance** - the percentage chance that the gene will “cross over”

**Return Value** - None

**Each member variable contains a get method.**

* 1. Class MasterGeneFactory
     1. Member Variables

None

* + 1. Member Functions
       1. Function *createMasterGene*

**Actions Performed** – creates a MasterGene object using the data in the provided data file

**Arguments** – None

**Return Value** – a shared\_ptr to the MasterGene object

* + - 1. Function *hasNext*

**Actions Performed/Return Value** – returns true if the data file has more master gene data which can be used by ::createMasterGene ; false otherwise

**Arguments** – None

* + - 1. Function *getInstance*

**Actions Performed/Return Value** – returns a pointer to a static instance of the MasterGeneFactory class. This is the only way to access this class, as it implements the singleton pattern

**Arguments** - None

* 1. Class Chromosome
     1. Member Variables
* **\_genes** – a vector of Gene objects which make up the Chromosome
  + 1. Member Functions
       1. Function *addGene*

**Actions Performed** - adds a Gene to the chromosome’s vector of genes

**Arguments** –

**Gene gene** - the Gene to add

**Return Value** – void

* + - 1. Function *begin*

**Actions Performed/Return Value** – Returns a ChromosomeIterator positioned at the start of the chromosome’s gene sequence

**Arguments** – None

* + - 1. Function *end*

**Actions Performed/Return Value** – Returns a ChromosomeIterator positioned one element past the end of the chromosome’s gene sequence. This is used for bounds checking in loops.

**Arguments** – None

* 1. Class ChromosomeIterator
     1. Member Variables
* **\_it** – a const\_iterator to the underlying vector of Gene objects
  + 1. Member Functions
       1. Function *operator++*

**Actions Performed** – Moves the position to the next Gene pair

**Arguments** – None

**Return Value** – None

* + - 1. Function *operator!=*

**Actions Performed** – Performs a Boolean comparison between two ChromosomeIterator objects

**Arguments** –

**const ChromosomeIterator& rhs** – the other iterator to compare against

**Return Value** – returns true if the iterators are not at the same position in the gene sequence; false otherwise

* + - 1. Function *operator\**

**Actions Performed** – Returns the symbol of a randomly chosen allele from the gene pair

**Arguments** – None

**Return Value** – a char representing the symbol of an allele

* 1. Class ChromosomeFactory
     1. Member Variables

None

* + 1. Member Functions
       1. Function *getInstance*

**Actions Performed/Return Value** - returns a pointer to a static instance of the ChromosomeFactory class. This is the only way to access this class, as it implements the singleton pattern

**Arguments** – None

* + - 1. Function *createChromosome*

**Actions Performed** – creates a Chromosome object using the supplied arguments and then populates its genes using GeneFactory

**Arguments** –

**string s1** – a strand from one parent organism

**string s2** – a strand from the other parent organism

**Return Value** – returns a Chromosome object