**CS105 Exemption Project 2018-19**

**Abinmon Jain**

**Reg No. 201834281  
Bsc. Software Engineering**

***Except where explicitly marked to the contrary by quotation or reference, what follows is all my own work. I have not knowingly allowed others to copy my work.***

**Abinmon Jain**

**22/03/2019**

**The fox and rabbit project initially contains the following classes** : An Animal Class, A Counter Class, A Field Class, A FieldStats Class, A FnRMain Class, A Fox Class, A Location Class, A Rabbit Class, A Randomizer Class, A Simulator Class and A SimulatorView Class.

The Animal class is an abstract class responsible for providing the methods that both the foxes and rabbits have in common. When both the rabbit and the foxes use the extends keyword, they inherit the methods and the variable from the animal class. The animal class has an abstract ‘act’ method which is inherited by the fox and rabbit class and they implement the method in their own class accordingly.

The Counter class provides methods that enable us to keep a count of each animal in the simulation.

The Field Class is responsible for creating the rectangular field in the simulation. It enables us to allocate an animal to a location on the field. It also contains methods that can be used to see what animal is in a certain location and whether or not the location is free or has any adjacent locations which are free. Both of which are used in the fox and rabbit classes.

The FieldStats Class is responsible for keeping count of all the different animals in the simulation and keeping the details of the population in a variable.

The FnRMain Class is the class that contains the main method to run the simulation.

The fox class is the one of the two main animal class that inherits methods from the animal class including the abstract act method which is implemented here. The act method calls the increment age and hunger method both of which increment/decrement the corresponding values by one. If it hasn’t died of starvation or old age, then it had the chance to give birth to new foxes in adjacent locations. It also has the chance to find food from an adjacent location but if there is no free locations in the adjacent cells then it dies. The find food method checks the cells nearby to see if there are any rabbits around and if there are and the rabbit is alive then it gets killed and eaten and it set the foxes food level to the rabbit’s food value. The give birth method again looks for adjacent locations which are free, if there is a free location then that location is removed from the shuffled list and a new fox is placed there. This method also checks if they can breed (i.e they are above the breeding age) and generates a random double and if this double is less that the breeding probability then the fox can give birth to a litter of size less than or equal to the max litter.

The Location Class is used to tell us where in the field a particular animal is located.

The rabbit class is the other main animal class that also inherits methods from the animal class by again using the extends keyword and it needs to implement the act method. However, in this class the act method on calls the increment age method. But if it is alive it can give birth to new rabbits and if there are any free adjacent locations then they can move there and if there isn’t then they die due to overcrowding. The give birth method is similar to the fox class where if they can breed and the generated random double is less than or equal to the breeding probability then they give birth to a litter of size less than or equal to the max litter.

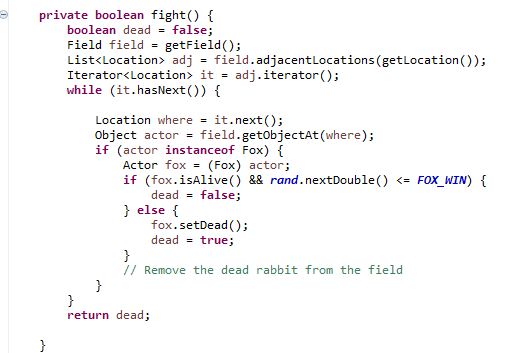
The Randomizer Class is used to generate random instances throughout the project having this enables us not to create a new random object every time.

The Simulator Class is responsible for setting the colour of the classes in the simulator. It also stored the animals as an ArrayList which is then used in the simulateOneStep method where new animals is added to the list after the act method has been called. This class also calls the SimulatorView class and uses a variable to reference it. The populate method clears the field and randomly fills the field with foxes and rabbits with a random age and location. It also has a step field which keeps count of the number of steps that has been passed.

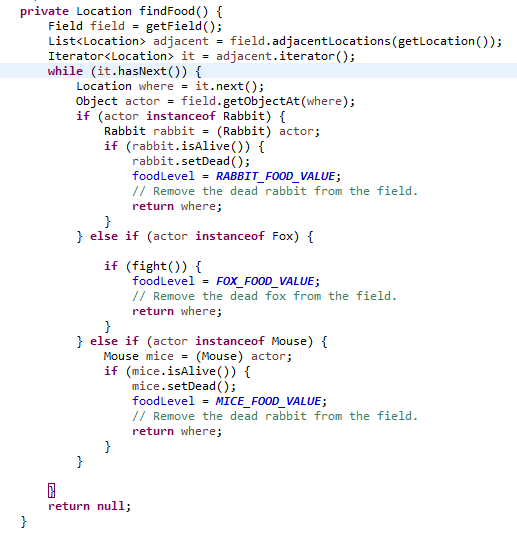
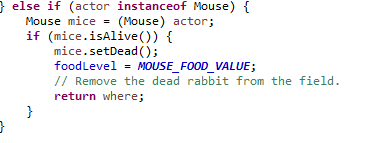
And finally, we have the SimulatorView Class which has a reference to a FieldView class as well as a FieldStats class. It keeps track of the population size with a variable called population. It also has a show status method which draws the fox and rabbit simulator.

**The changes I made to the project include:**

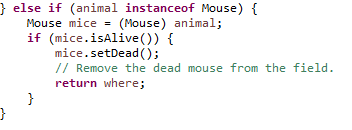
Firstly, I created two new animal classes, the Lynx and the Mouse class which both are like the fox and rabbit class respectively. I had implemented the Lynx class so that it kills and eats all the other animals (i.e foxes, rabbits and mice). However, I have also added a method to this Lynx class which fights with foxes in adjacent cells and it dies if the random double generated in the fight method is less than the probability of the fox winning. It also gets killed and eaten by the hunter class.



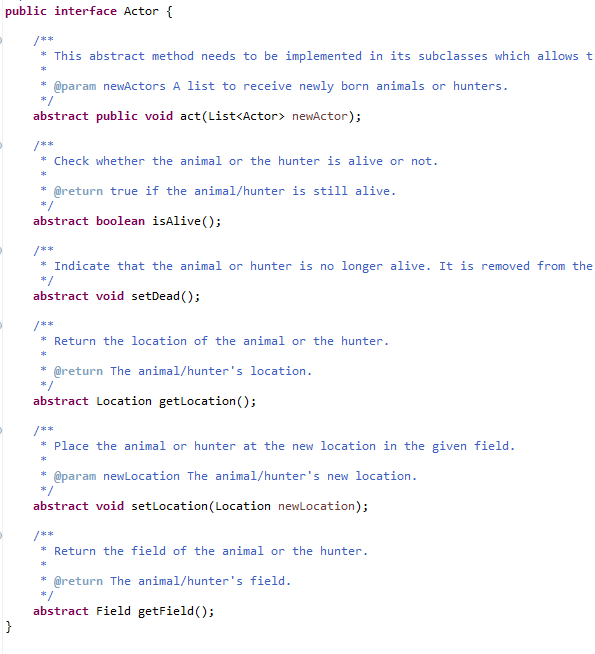
Again in the find food method is similar to that of the foxes class however this fight method is also called in the find food method such that if a fox loses the fight the lynx would just eat the fox that has died.

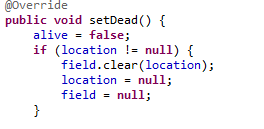


All the other methods are like that of the fox’s class. I have also added the mouse to be part of the food that the fox eats in its class and it would set the food value to the value of the mice:

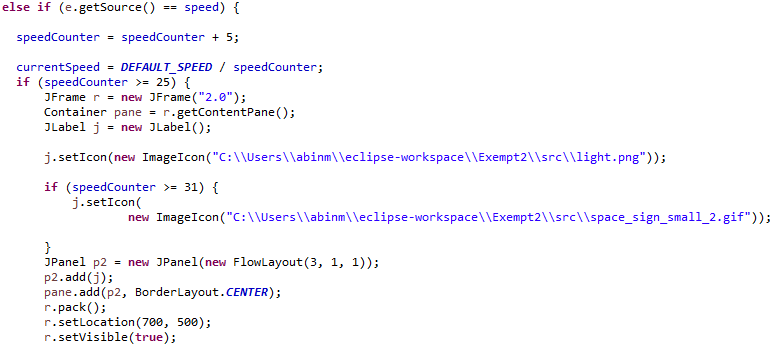
The mouse class is the same as the rabbit class and it gets eaten by both the lynx and the fox but even though it gets killed by the hunter class it does not get eaten by the hunter. The following code is the code for the mouse in the hunter class:

I have also added an Actor Interface which is superclass of both the hunter class and the animal class. The interface contains abstract methods both of which are common in the animal and the hunter class. Then both the classes inherit the actor interface and they implement the methods accordingly. An example of the implemented method:



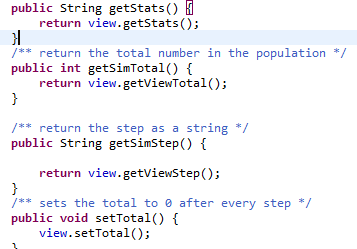
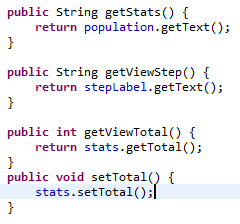


This section shows how the speed is changed, each time the speed button is pressed then, the variable increments by 5 and the variable is the divisor for the static value which is then passed into the setDelay() method, as the value decreases means that there will be more ticks/second hence increasing the speed

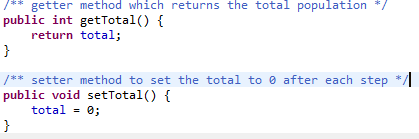


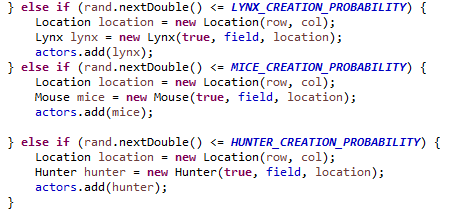
I have also added a getter method to the simulator and SimulatorView class in order to access the data about the current population as well as return the total and the step number, which is used in the Gui to store it into a text file. **Simulator Class Methods:**

**SimulatorView Methods:**

**In fieldStats class I made a new variable**

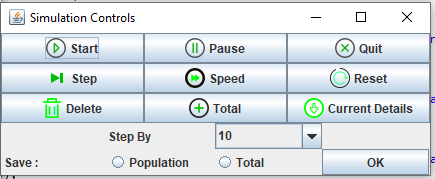
 **and used getter and setter method:**



Also, in the simulator I had changed the variables to do with Animal to Actor

so that the simulator can also use the methods for the Hunter class, I have also added the 3 classes I made, to the populate method.

And finally, I have a made a class for my GUI that implemented the ActionListener Interface and called a Simulator Object and initialized it, which allowed me to control the object using buttons. I had made a frame to which I add my buttons. The buttons I added include: Start, Pause, Step, Reset, Speed, Quit, Delete, Current Details, Save and a JCombobox(Drop-down list) which allows you to step by a certain amount.

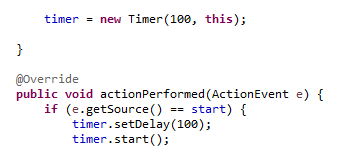


I had created and initialized the button along with an image added to the button using the Image Icon, I had also added an ActionListener to corresponding button which was then used in the actionPerformed method to carry out the required function.



Similarly, I had also made the JComboBox in this method and added all the buttons and the ComboBox to the JPanel and added the panel to the frame.

I had used the Timer class in order to control the start, pause and reset button. The timer object was declared and initialized in the method that made the frame, but the timer was only started when the start button was pressed. The timer object came with the action listener already added when it was initialized. In the actionPerformed method, when the timer starts, every tick causes the simulator to step once.

I 

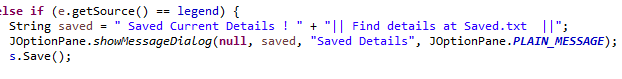
The JComboBox worked similar to that of the button, the value which was chosen by the user was converted to a string and the string was compared and the simulate(int n) method was called.

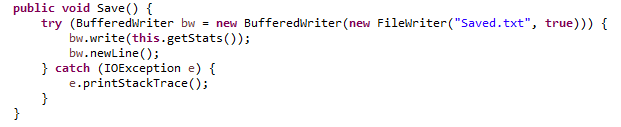


When the user clicked the current details button a dialogue box was shown with the current details of the population.

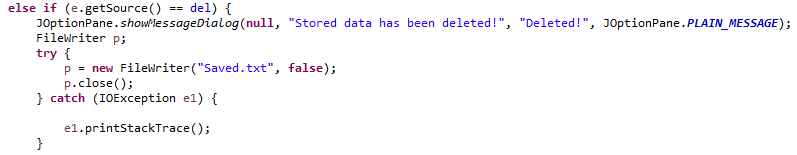


And finally, the save and the delete buttons again displayed a message saying either the data was saved or deleted and both contained a buffered writer a file write class that wrote the details of the population into a text file or deleted the data from the text file. The save method was implemented in the simulator and called when this button was pressed.

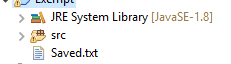
Save:



**Delete:**



***Saved File Location:***



***Lessons***

My Exemption Project includes the following lessons which I have learned throughout the first year of the course. They are:

* I have created objects, methods, and declared and initialized variables. I have used methods which are of both having a return type or being void which also include the use of accessor and mutator methods.
* I have made constructors and methods which take in parameters,
* I have used basic java primitive types and object types,
* I have used if and else statements,
* I have used modularization techniques to divide the project into smaller sections making it easier to work on.,
* I have also used the concept of calling methods inside constructors as well as inside other methods.
* I have used flexible sized collections (Array Lists) to store objects in variables.
* I have used loops (for loops) to iterate over a collection and return the desired objects as well as using these loops to search a collection.
* I have used the Java 8 API library to look up different classes I could use and their methods and how to implement these methods,
* Even though the Random Number generator Class was provided I had used this class to generate random numbers for my probabilities in my animal classes
* I have also used class constants for probabilities of the animal class,
* I have used inheritance hierarchy to define an abstract class which is then implemented according in the sub classes.
* I have also used the concept of casting,
* I have also used the concept of using a wrapper class when saving and deleted the stored data from the text file,
* I have used try and catch to catch any FileNotFound exceptions thrown when trying to write to a file,
* I have also used the concept of overriding methods that has been passed down to the subclass
* I have also added A Gui Class which controls the simulator, I have also implemented the ActionListener class which allowed me to control the simulator using buttons
* Again, using the Java 8 API I was able to look up the different features that is available to me for my Gui.