Lab sheet 3: More on Inheritance

Setter: Idris Skloul Ibrahim

Attempt Task 3A in the tutorial and finish Task 3B in the lab. Don't forget to commit your code before you leave the lab!

Please use the code in `lab3/foxesAndRabbits2` for the following exercises.

Note: This lab is worth double points (10 Marks)!

Deadline:

Lab 3 should be checked off by me or a lab helper by Friday 01/04/2016.

Complete Assessment Report:

The following assessment questions are based upon your FoxesAndRabbits code from Lab 3 ("More on Inheritance"). In order to answer the questions you must have completed all of Task 3A and 3B on the lab sheet. Each answer is worth 2.0 points, and 2.0 Marks for code.

Please write a report 1 - 2 pages, and between 30 - 60 words per question.

Assessment Questions:

Q1. By introducing the Animal class in version2 of the code, the simulateOneStep method in Simulator.java has changed. Explain how and why this version is better. Explain in terms of coupling, cohesion and code duplication.

Q2. Explain how you improved the code by introducing the PopulationGenerator class in Task3A.1.

Q3. Explain your newly introduced animal (Task 3B.1) in terms of features and behaviour. Explain what is differentiates the new animal from the existing animals, i.e. how much new code was written/ modified?

Q4. Describe the new superclass you have introduced and how it relates to the other classes via inheritance. What methods did you have to move? Comment on the advantages and disadvantages of your current implementation as Abstract Class.

Task 3A: Refactor the Code

3A.1 Population Generator

Look at the previous version of the code in: **`lab3/foxesAndRabbits1`** and note the improvements made by introducing the Animal class.

Also note the changes made to the **`simulateOneStep`** method in **`Simulator.java`**: By introducing the Animal superclass we have removed the dependencies (couplings) of the **`simulateOneStep`** method and the Fox and Rabbit class. The Simulator class, however, is still coupled to Fox and Rabbit in the populate method.

<u>Your task</u>is to further improve this code by introducing a new class **`PopulationGenerator`** and move the populate method into this class.

Now only this new class is coupled to the concrete animal classes, making it easier to find places where changes are necessary when the application is extended. The **`PopulationGenerator`** should be called in **`Simulator.java`**.

3A.2 Animal Behaviour: Ageing and Breeding

- Move the **`age`** field from Fox to Animal. Initialize it to zero in the constructor. Provide accessor and mutator methods and use these in Fox and Rabbit (rather than direct access to the filed).

- Move the **`canBreed`** method to Animal. Provide a method **`getBreedingAge`** in Fox and Rabbit and use it in your new **`canBreed`** method.

- Move the **`incrementAge`** method to Animal by providing an abstract **`getMaxAge`** method in Animal and concrete versions in Fox and Rabbit.

- Move the **`breed`** method to Animal. (Think how best to do that!)

Task 3B: Introduce New Animals

3B.1 Define a new type of animal

- Define a completely new type of animal, as a subclass of **`Animal`**. You will need to decide what sort of impact its existence will have on the existing animal types. For instance, your animal might compete with foxes as a predator on the rabbit population., or your animal might prey on foxes but not on rabbits.

- You will probably find that you will need to experiment with the configuration settings.

- Don't forget to modify the **`populate`** method.

- You should also define a new color for you new animal class in **`SimulatorView`**. You can find a list of pre-defined colour names on the API page documenting the **`Color`** class in the **`java.awt`** package.

3B.1 Define a new Superclass

Introduce a new Abstract Class, which helps to reduce code duplication within other animals, for example, you could introduce a superclass **`Predator`**, which implements a hunt method.