F27SB Software Development 2

OOP Coursework 3: Foxes and Rabbits

Refactor the Code

Population Generator

In the lecture, we created the **`Animal`** class for **`FoxesAndRabbits`**. If you are looking at this coursework before we had this lecture, you can compare the code in the Coursework repository with the code in the Live Sessions repository.
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.

Your task is to further improve **`FoxesAndRabbits`** by introducing a new class **`PopulationGenerator`** and move the **`populate`** method from the **`Simulator.java`** into this new class. You might also have to move some of the variables and fix the resulting errors in other classes to now use the **`PopulationGenerator`** instead.

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

(Created new class: 0.5P,
Moved method to new class: 0.5P,
Simulator is using new class without errors: 1P)

Introduce New Animals

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 it will have on the existing animal types. For instance, your animal might compete with foxes as a predator on the rabbit population, or it might be another prey animal. You can also try to create an apex predator that preys on everything

including foxes but that introduces additional difficulty as suddenly some animals can be both predator and prey. Hence, we only recommend this if you feel confident enough in your programming abilities.
- You will probably need to experiment with the configuration settings (the constants in the animal subclasses, e.g. breeding age/probability, litter size, max age, etc.) in order to re-establish the equilibrium in your ecosystem.
- Don't forget to modify the **`populate`** method.
- You should also double check the **`giveBirth`** method. A common copy and paste error is that your new animal gives birth to foxes or rabbits because you copied the class and forgot to change that.
- You should also define a new colour for your new animal class in the constructor of the **`Simulator`** class. You can find a list of pre-defined colour names on the API page documenting the **`Color`** class in the **`java.awt`** package.

(Created new Animal: 0.5P,
Simulation uses new Animal: 0.5P)

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 replacing the `findFood` method in `Fox`. For this to work you might have to also create a superclass **`Prey`** if you do not want to force your `Predator`s to only eat `Rabbit`s.
Hint: If you chose to create an apex predator, this part will be more complex as you will need to use multiple inheritance. You could try and use interfaces instead of an abstract class.

(Created and used superclass: 1P)

(Optional) Moving fields and methods

As an additional challenge, which fields and methods could be moved from the subclasses `Fox` and `Rabbit` to `Animal`, `Prey`, or `Predator`? Move as many fields and functions as possible to reduce code duplication. Think about what would make sense from a biological point of view, e.g. do all animals have an age and can give birth?

Marking

In each coursework, you can achieve a total of 4 points. Each question awards different amounts of points for different parts of the question. Partial completion of a task will award the partial points listed underneath. Most descriptors of points are self-explanatory in context of the task. Where further clarification is needed, you can find that below.

Your work will be checked by a lab helper during your assigned lab slot. Once all tasks are checked, the points will be used to calculate your marks. Please, understand that the lab helpers are not marking your work but are checking the completion of subtasks. As part of this check, you will need to explain how you solved the given task. **Only successfully completed sub-tasks will award points.**

The marks will be released on Canvas after the marking deadline has passed. This is not an automatic process so please, be patient. Once the marks are released, you will be notified via Canvas. Please, **make sure to check your marks as soon as possible**. If there are any issues, please contact your teaching team immediately.

Collusion and Plagiarism

As mentioned above, you will need to explain your work during the demo session in the lab. If you are not able to explain how you arrived at the solution to the task, we need to assume that you did not do the work yourself. We do, however, know that you might be anxious or nervous during the session. Please, rest assured that this is not an interrogation and you can take all the time you need to explain your solution.

If there is reasonable doubt that you solved the given problems yourself, you will not get any points for this task. If there are concrete indications that you copied your answer or colluded with other students, we might also start an official investigation.

Please, make sure to fill and sign the Declaration of Student Authorship form for each coursework and upload it to Canvas. If you do not upload the form, we will not be able to give you any marks for this coursework.

If you feel unjustly accused of plagiarism or collusion, please contact your teaching team.

Coursework submission

Unless stated otherwise, all code parts of your work need to be committed and pushed to your GitLab fork. You need to upload the Declaration of Student Authorship to Canvas, and you need to present your solution to a lab helper. If you fail to do any one of these steps, you will not be awarded any marks.

The deadline for submission can be seen on Canvas. You will need to **present your work to a lab helper any time before the deadline during your allocated lab slot**. If you do not manage to present your work before the deadline, you can do so at the first lab after the deadline but will incur a 30% late penalty. If this late submission was caused by issues out with your control, you can apply for mitigating circumstances to have this late penalty removed.

If you also fail to present your work at the lab following the deadline, we will not be able to give you any marks for your work. Similarly, if this was caused by circumstances out with your control, you should apply for Mitigating Circumstances.
Please, note that we are not allowed to give individual extensions. If you cannot submit your work on time, you will need to apply for Mitigating Circumstances.