**Foxes and Rabbits - FAQ**

**In the Field class, what does** depth **represent?**

The number of rows in the field; you could also refer to this value as height.

**In the Field class, why are some methods overloaded?**

Overloading refers to two methods with the same name and different parameter lists. Methods are overloaded to provide flexibility, allowing a programmer to call a method that functions the same way using what they know at the time.

**Why does the Field class initially store Object types?**

The initial state of the project has separate classes for Foxes and Rabbits, and Field has to store both. Without a super-class to unify them, the only type that is-a Fox *and* Rabbit is Object. Don't skip the refactoring - seeing a bad solution morph into a good solution using inheritance is important.

**How can I test that, for example, my Field class' methods are working?**

In a Tester class, create a smaller (e.g. 4x4) Field object. Create an animal at a particular Location and place the animal in the Field. Print the Field (which should have an overridden toString). Test moving and removing the animal and ensuring your adjacentLocations and freeAdjacentLocation methods return what they should.

**Why are all my foxes/rabbits dying on the second step of the simulation?**

Make sure all of your Field methods are working, e.g. move() and place(). See the "How can I test…?" question for tips on writing tests. It could be that they're not dying, they're being lost when moved.

**Why are all my foxes/rabbits dying after** N **steps?**

Assuming you haven't modified simulation parameters (e.g. Fox creation probability, Rabbit breeding probability, max life expectancy, etc.), make sure your animals are dying when they're supposed to. Also, if your animals are dying off after MAX\_AGE steps, your animal probably isn't breeding new animals.

**Why does my simulation look slightly different than the video?**

As long as your foxes and rabbits tend to move like the example, and the simulation usually remains viable for several hundred steps, don't worry about minor differences. Small implementation details can change how the simulation works.

**How do I make a random below one (for the fox/rabbit creation in the Simulator class)?**

Rather than multiplying the creation probabilities by 100 and using java.util.Random's nextInt() method, use the Random class' nextDouble() method, which returns a random floating point value 0 <= x < 1. (The static method Math.random() does the same thing.)

**Coming up with tests to debug this project is hard!**

Indeed, no one ever said computer science was easy. Also, that's not a question.