CSCI165 Computer Science II
Lab Assignment
Hobbits vs Nazgul

The goal for this programming project is to create a simple 2D predator–prey simulation implemented as a cellular automata. In this simulation, the prey are Hobbits, and the predators are Nazgul. These beings live in a fictional 2D world composed of a 30 X 30 grid of cells. Only one being may occupy a cell at a time and the Hobbits are constantly on the run from the voracious Nazgul who like nothing better than to feed on the plump, un-muscled, succulent Hobbit flesh. Around the edge of this world there is an ominous city wall. Due to the casting of some ancient black magic the Nazgul are not capable of crossing this wall. They are essentially repelled and must change direction. On the other hand the Hobbits, who appeased the witches with hand crafted ales and excellent long bottom leaf are capable of ascending these walls and tele-porting to the other side of town. When a Hobbit approaches a particular edge it wraps around to the opposite edge and appears there, similar to the mythological beast Pac Man. As this world is an automata, it does not require any human interaction. It is simply seeded with life and set on its inevitable path. As the seeding is random there are infinite scenarios that can play out. Time in this world is simulated in discreet steps which can be implemented as a loop iteration. A single iteration over the entire world is equivalent to a 24 hour Earth day. Each being performs some action every time step. While each species is of the same genetic lineage and share the same basic behaviors, the implementation is vastly different. The two basic behaviors are **Move** and **Breed.** Your task is to create an inheritance hierarchy that allows for these behaviors to be triggered polymorphically.

The Hobbits behave according to the following model:

- **Move:** Every time step, a Hobbit randomly tries to move up, down, left, or right. If the cell in the selected direction is occupied it should try another direction. If an adjoing cells are occupied then the Hobbit does not move
- **Breed:** If a Hobbit survives for three time steps, then at the end of the third time step (i.e., after moving), the Hobbit will breed asexually. This is simulated by creating a new Hobbit in an adjacent (up, down, left, or right) cell that is empty. If there is no empty cell available, no breeding occurs. Once an offspring is produced, that particular Hobbit cannot produce an offspring until three more time steps have elapsed.

The Nazgul behave according to the following model:

- **Move:** Every time step, if there is an adjacent cell (up, down, left, or right) occupied by a Hobbit, then the Nazgul will move to that cell, stab the Hobbit with its Ringwarith sword and consume the Hobbit. Otherwise, the Nazgul moves according to the same rules as the Hobbit. Note that Nazgul are not cannibalistic and if there is a Hobbit within edible range then that Hobbit **must** be consumed.
- **Breed:** If a Nazgul survives for eight time steps, then at the end of the time step, it will spawn off a new Nazgu in the same manner as the Hobbits.

- **Starve:** If a Nazgul has not eaten an Hobbit within the last three time steps, then at the end of the third time step, it will starve and die. The Nazgul should then be removed from the grid of cells.

During a single time step each being should move in the order they appear on the grid, starting at position (0, 0). The move and breed behaviors should be called without checking the class type. Checking the class type defeats the purpose of polymorphism and should only be done if you need to invoke a behavior that is not shared among the siblings. In order to achieve this here you can have the Nazgul method **move** call **starve**. Then when move is invoked polymorphically starve will also be invoked without having to perform a type check.

Write a program to implement this simulation and draw the world using ASCII characters of “o” for a Hobbit and “X” for a Nazgul. This should be accomplished with an over ridden **toString()**

Create an Abstract class named **Organism** that encapsulates basic data common to both Hobbits and Nazguls. This class should at least have the two abstract methods **move** and **breed.** Hopefully you are able to recognize that your world grid should be of type Organism. This will allow both Nazgul and Hobbits to populate the grid.

Important note: You can not instantiate an abstract class:

- **You can NOT do this: Organism o = new Organism();**
- **You CAN do this: Organism o = new Nazgul();**

You may need additional data structures to keep track of which beings have moved. Notice that I am leaving most of the design and implementation decisions up to you. The issues listed above are the **minimum requirements**, as is the implementation of the polymorphic methods move and breed. If you have additional creative ideas that go beyond what is listed here please pursue them. You do not need to ask for permission, just make sure that there is documentation. **Example**: It would be trivial to add additional organisms here. All you need to do is define the **toString, move** and **breed** behaviors and plug them in

Initialize the world with 10 Nazgul and 150 Hobbits at random locations. Each time the simulation is executed it should begin with a different configuration of beings. You should see a cyclical pattern between the population of predators and prey, although random perturbations may lead to the elimination of one or both species. If you would like to include an arbitrary hesitation between loop iterations you can execute **Thread.sleep(millisecond_Value);**

Here is an example of how the world should be drawn. The city wall is over emphasized and you do not need to worry about making it so bold. You have the option of whether or not you wish to draw the actual grid lines. Be creative.

```
X     o o o o o o           o o o o o o o   o o
o o       X       o   X     o o o o         o     X   o
  o o                       o o o     o     o         o
    o o       o   o o o o o o o o     o   X o         o
      o     o               X     o o o o o o           o
o o o o o         o o o o o o o o o o     o       X o o o
        o o o o   X                   X   o     o o o o o
        o o o o o           o o o o o o o o o o o o
        o o o o                 o o o o o o o o o o

X     o o o o o o           o o o o o o o   o o
  o o o o o o o o o o o o o o o o     X     o o o o o o
o o       X       o   X     o o o o         o     X   o
  o o                       o o o     o     o         o
    o o       o   o o o o o o o o     o   X o         o
      o     o               X     o o o o o o           o
o o o o o         o o o o o o o o o o     o       X o o o
        o o o o   X                   X   o     o o o o o
        o o o o o           o o o o o o o o o o o o
        o o o o                 o o o o o o o o o o
    o   o o o o o o o o     o   X o         o
  o               X     o o o o o o           o
        o o o o o o o o o o     o       X o o o
o o o   X                   X   o     o o o o o
```