∗ 3.1 Assignment 3: Lemmings

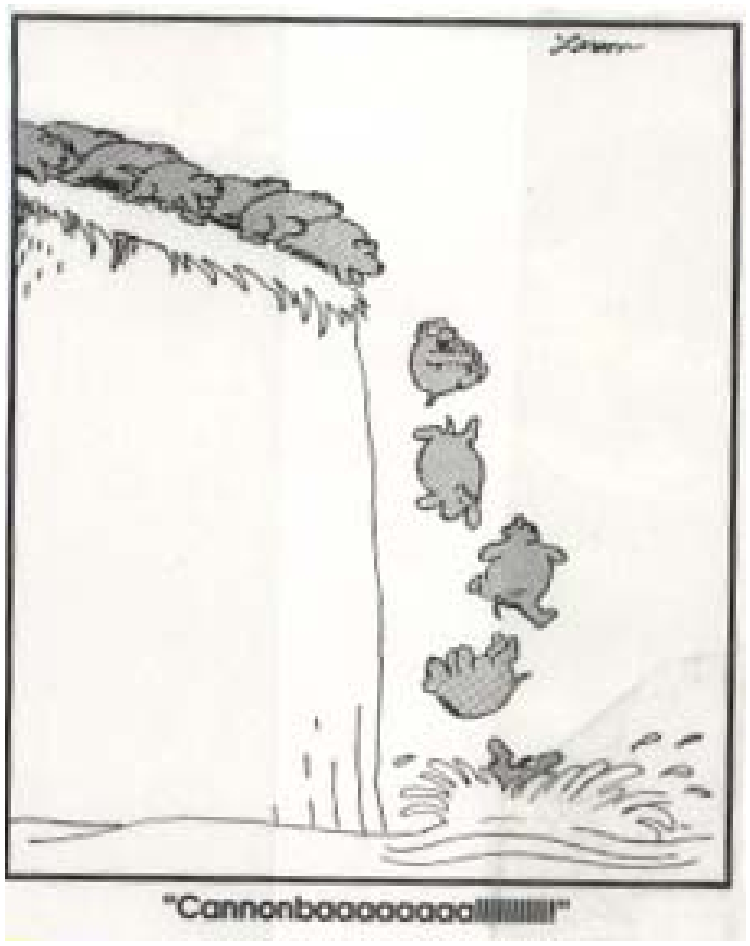

Figure 3.1: The myth of the lemming.

For this assignment, you will have to make a simulation in which threads, concurrency, serialization and sockets will occur.

∗ 3.1.1 Background information

Lemmings are small rodents that mostly live around the Arctic. Historically lemmings are known for their strange migratory behavior, which many considered to be mass-suicide. Whenever the population of lemmings becomes too dense in an area, a large group of lemmings will start to migrate. During this migration, the lemmings may run into a very wide space of water that they will try to cross. Many will drown, leading to the idea of mass-suicide. This idea of mass-suicide led to many different stories and even the well-known video game.

For this assignment, you will try to simulate this behavior. You will have to implement lemmings that can while they live, move, birth children, sleep and sometimes commit suicide.

One main problem that you will probably run into is preventing and solving *deadlocks* and *race conditions*. In order to find this, you can set breakpoints in your IDE for debugging. You can do this by double-click the area left of the statement for which you want to place this breakpoint. This gives you the opportunity to check the variables on the current stack.

∗ 3.1.2 Fields

The world of lemmings consists of multiple fields. Each field is considered to be an independently running application, meaning that you will have to start up each field separately. Every field should also have a server-like behavior. This means that it should listen to a certain port, accept incoming connections and handle these.[1] These steps will help you create such a field:

1. Create a class Field. This class should start in a separate thread. In order to do this, either use the class Thread or interface Runnable. A field should have a ServerSocket and this ServerSocket should have a port to which it should listen. Please note that this port should be different for each field running on the same machine.

2. Make sure that, when the thread has been started, it will listen to the ServerSocket for incoming connections and accepts these.

3. This incoming connection should be handled in a different thread so that the server can continue listening to new incoming connections. In order to do this, create a new class InputHandler. Make sure that when this class is started as a new thread, it will read an object from the accepted connection by the use of an ObjectInputStream. To test if this works correctly, make the InputHandler print the object.

4. A class FieldView is given. Add this class as a view to the class Field. To make sure that it will compile, you will also have to add some methods to Field. Look at the update() method in FieldView and implement the methods to retrieve

   - a listing of all lemmings in the field. Since there are no lemmings in the field, return ‘<no lemmings>’ for now;
   - a listing of all known fields. Since there are also no known fields, return ‘<no fields>’;
   - the number of lemmings in the field. Since there are no lemmings in the field yet, return 0;
   - the capacity of the field;
   - the address on which the server can be reached;
   - the port the server is using.

5. The view should function according to the Model-View-Controller pattern.

∗ 3.1.3 Lemming

By now, we have created two classes that function as a thread and combining these two classes has resulted in a server. The next step is to send an object to the server. This will eventually be used to send lemmings between different fields, allowing them to move. Before an object can be sent, some preparation has to be done, better known as marshalling or serialization. In order to do this, you will have to implement the Serializable interface. Every object that has to be sent has to implement this interface and the objects can then be sent by the use of an ObjectOutputStream. In steps:

7. First, we will create a class FieldConnector which will function as a client. Make sure that this FieldConnector knows everything about the location of the server, meaning that it will have a String with the address and an Integer with the port. If all the fields run on the same machine, you can use ‘localhost’ as address. The port should be the same integer that you used for the server.

[1]The tutorials of sun might be of use here. You can find them for example on http://java.sun.com/docs/books/tutorial/networking/sockets/clientServer.html

8. Create a new class Lemming. This class should also be a thread even though it is at this moment not yet necessary. Make sure that the class implements Serializable since it should be sent towards the server.

9. FieldConnnector should have a method send(Lemming l) in order to send a lemming. Note that the FieldConnector does not have to be a thread. The send method should eventually be called by a Lemming-thread when it wants to move. First, let the method create a Socket. Secondly, it should create an ObjectOutputStream with the OutputStream of the socket and use this in order to send the lemming.

10. Make sure that the lemming arrives at the server.

∗ 3.1.4 Neighboring fields

By now, we have created two classes representing a field, namely Field and FieldConnector. We will use these to create a network of fields. For this, a Field is a node in the network and every Field should keep track of a list of FieldConnectors. This way, the connections between the nodes are made and we have created a network. In steps:

11. Create a class FieldMap. This class keeps track of a 'map' with all the addresses and ports that a Field knows. These maps should also be passed between fields whenever a lemming moves, meaning that this map should also be Serializable.

12. Give Field a FieldMap attribute in order to keep track of all known neighboring fields.

13. Also give Lemming a FieldMap attribute. This map can then be used by the lemming to move from field to field.
    - Make sure that when a lemming moves, it first updates its FieldMap to the map of the current field.
    - Also make sure that when a lemming arrives in a new field, it updates the FieldMap of that field by adding each Field known by the lemming but not by the field.

∗ 3.1.5 The lifeline of a lemming

Now that we have a network of fields, the fields are ready to be inhabited by lemmings. Before we continue with this, first some more information is necessary about the behavior of lemmings. A lemming will continuously execute a certain activity until it dies. This activity can be:

- A lemming can sleep for a random time period, with a maximum of about 2-3 seconds;
- A lemming can give birth;
- A lemming can move to another field.

You are allowed to influence the chances of each action. Another activity that a lemming does is committing suicide. This happens whenever a field gets overcrowded or when there is not enough to eat. The number of lemmings that can live in a field is always somewhere in the order of 10. This is a constant of a field, meaning that it will not change over time. Different fields however can have a different capacity. When a field gets too crowded, a lemming will commit suicide according to the following rules:

- A lemming commits suicide when it moves to a field that is full;
- When a field is full and a lemming gives birth, the parent will commit suicide in order for the child to live.

Note that a field should never have more inhabitants than the maximum capacity allows. In steps:

14. Make sure that a lemming is a thread that continuously chooses an action. Start by implementing sleeping, since this is the easiest, next implement giving birth.

15. Now implement movement between fields. In order to do this, you probably have to change Field and InputHandler to accept a lemming and start it in the other field. The lemming also has to die in the old field.

∗ 3.1.6 Bonus

The following can help you to get a higher grade. Keep in mind that it is of course not possible to receive a higher grade than a 10.

- Implement a gender for the lemmings. First start by creating subclasses of Lemming and secondly consider the logic about giving birth. When a lemming (l1) decides to make a baby, it first have to finds another lemming (l2) of a different gender that always wants to have a baby. Whenever this is the case, l1 alerts l2. The female lemming will then give birth and the male will continue with its life. Whenever there is no lemming l2 waiting for a partner, l1 goes into waiting.[2]

- Often lemmings will die, while they still wanted to do so much with their life. This is why they leave a will and you will have to implement this. Make sure that every lemming that dies, leaves a certain message. Finally, all these wills will be printed on a server that represents a Graveyard. In contradiction to fields, lemmings can not live on a Graveyard. Make sure that each Graveyard also has a GraveyardConnector. Use the given GUI as a view for the Graveyard. Note that also a view has been given for the Graveyard.

- It might be possible that a Field is disconnected. Whenever this happens, it is possible that invalid addresses occur in the FieldMap. Make sure that whenever an address is unreachable, it is removed from the FieldMap.

[2] A tutorial about sleeping and waking a thread can be found on: http://docs.oracle.com/javase/tutorial/essential/concurrency/syncmeth.html