Dying cats

Introduction

Schrödinger's cat is a thought experiment, sometimes described as a paradox, devised by Austrian physicist Erwin Schrödinger in 1935.

Here we present a variation of Schrödinger's cat, that will be simulated with robot cats. Your task is to create a small application that uses networking to achieve the 'teleportation of cats'.

Description

There are a fixed number of boxes, each with a predetermined capacity. Each box acts as a separate, independently running application. There is an initial number of robot cats, randomly distributed in the boxes.

A box will trigger its killing mechanism from time to time, killing (breaking) all the robot cats inside. These broken cats can not move anymore and have to be removed from the box by the manager.

Cats can randomly teleport to one of the other boxes. The teleportation can be successful or it can fail. It will fail if the box is completely full, or if the teleportation encounters technical issues (such as network failure, serialization issues, etc.). If this happens, the cat simply disappears in limbo.

A cat has 9 lives and each teleportation event takes up one of its lives. When a cat has 0 lives, it turns off (dies/breaks) and can not move from that box until the manager removes it. Only cats that have more than 0 lives left, can be revived back up to 9 lives by the manager.

There are 3 actions that a cat can choose from:

1. Purr (sleep)
2. Teleport
3. Clone itself

There is a manager (Schrödinger) that has the task to open any box and fix the cats by recharging them as well as removing the broken ones as these cannot be fixed anymore. Moreover, the manager can randomly decide to add a few new cats back into the box.

Guidelines

This application should be based on the following components, that you will design and develop:

- Box - this class should accept incoming requests and either welcome or reject the cats based on the capacity of the Box. Make sure that a Box is running as a thread.
- Cat - each Cat has 9 lives at its creation. Since they need to choose between an action during their lifetime, they should be implemented as threads that can be paused and continued.
- BoxManager - this is the manager class, that can contact and inspect the content of any of the Boxes. It should be able to reset the recharged Cats (restart their thread) as well as randomly decide on spawning Cats into the currently inspected Box.
- Teleportation of cats - Each Box should act as a server by using a ServerSocket that listens for incoming connection requests. Cats will use Sockets to connect to the Boxes. Make sure that each Box uses a ConnectionHandler to deal with multiple consecutive requests, to avoid blocking.

Requirements

The list of requirements for this assignment is as follows:

1. Cats, Boxes and BoxManagers have thread functionality.
2. Boxes should be implemented as individual applications.
3. To ensure that a cat arrives at its destination, use the appropriate protocol.

4. Proper starting, pausing and killing of threads.
5. Because multiple threads printing to one console is difficult to track , make a **simple** UI: A frame displaying print statements, used for the boxes and the manager (and perhaps the cats). *Do not spend any more time than necessary on your UI. Do not make a fancy editor. Do not add actions. Only a simple view is sufficient.*

Some useful resources: Server-Client pair , Server Socket , Socket , ObjectOutputStream , ObjectInputStream , Thread , Synchronized Methods , Random .

Bonus

You could score some bonus points by implementing some of the following features:

- The manager can also introduce new boxes into his experiment, while the experiment is already running. Cats should be able to teleport to this new box as well.
- Extend the behavior for the robot cats by adding the tracking of battery resources. Thus, cats will lose battery life based on each action they choose. So sleeping affects it the least, then teleportation and lastly cloning halves its battery life.