Concurrent Objects (Monitors)

Homework 4

Alex Lindberg
alex5@kth.se

February 25, 2022

Introduction

This is a short report detailing the fourth homework — *Programming with Concurrent Objects (Monitors)*. Implementations for task **[1,5,6,7]** are included. All implementations were done in Java using monitors and synchronized methods.

1. Fuel Station

This implementation uses a monitor `Station` and two runnables, `Vehicle` and `Supplier`. Each Vehicle repeats the pattern of `requestFuel`, `refuel` and `releaseFuel`, which are synchronized methods and part of a `Station` object. The vehicles need to check for a few conditions, namely if there's currently a supplier refilling the station, if there's an open docking spot and if there's enough fuel.

```
public synchronized void requestFuel(int cap)
    throws InterruptedException {
    while (nDepositing > 0
        || maxDocks < nDocked
        || nNitrogen + nQuantum <= cap)
        wait();
    nDocked++;
}
```

It then refuels with whichever fuel there's enough of and even mixing if it has to. After refueling it notifies any waiting vehicles to proceed.

Similarly the `Supplier` waits for any current deposits to be finished and for there to be space in the fuel tank for whichever fuel the supplier is carrying. Then it deposits and calls `requestFuel` for itself.

Below is a sample of the output.

```
Vehicle 3> refuling, Supply[N:40, Q:20]
Vehicle 9> refuling, Supply[N:20, Q:20]
Vehicle 4> 4 trips left
Vehicle 11> 4 trips left
Supplier 1> Depositing, new Supply[N:20, Q:100]
Vehicle 6> 4 trips left
Vehicle 8> 4 trips left
Supplier 1> refuling, Supply[N:20, Q:100]
Vehicle 0> refuling, Supply[N:20, Q:80]
```

5. Hungry Birds

The implementation uses two sub-classes, `ParentBird` and `BabyBird`, which share a resource `worms`. The sub-classes are runnable, meaning we just need to provide an instance of these classes to a thread which will run it for us.

We use two synchronized methods to access the `Dish` of worms. The parent calls `fill` to refill the dish, and the children call `eat` to consume worms. When a child calls `eat` and the dish is empty it will alert the parent but it won't stop "chirping" until the dish is refilled.

The implemented solution should be fair. The consumers (baby birds) can't consume food if there is none, and the parent won't refill if there is. The consumer that finishes the food supply notifies the producer to refill.

```
Bird 2: !!!*CHIRP*!!!
Parent> Refilled with 5 worms
Bird 1> *slurp* nom (4)
Bird 3> *slurp* nom (3)
Bird 4> *slurp* nom (2)
Bird 5> *slurp* nom (1)
Bird 2> *slurp* nom (0)
Bird 4: !!!*CHIRP*!!!
Bird 5: !!!*CHIRP*!!!
Bird 3: !!!*CHIRP*!!!
Bird 1: !!!*CHIRP*!!!
Parent> Refilled with 5 worms
```

6. The Bear and Honeybees

This solution is almost the same as the Hungry Birds solution. The exception is that the honeybees will stop and wait to be notified once the pot is full. But other than that it works the same.

Below is a sample of the output.

```
Bee 3> ~Bzz~ *spits honey*
Bee 4> ~Bzz~ *spits honey*
Bee 3> ~Bzz~ *spits honey*
Bee 3> ~BzzZzZZZZz?!~
Bear> *yawn* Snacc time :)
Bear> *yawn* sleepy
Bee 1> ~Bzz~ *spits honey*
Bee 2> ~Bzz~ *spits honey*
```

7. One-Lane Bridge

This implementation uses four operations to manage crossings. For each direction, north and south, there's an `Enter` operation and an `Exit` operation. Entering and exiting from the north is done in the following way.

```
    while (nSouth > 0)
        wait();
    ++nNorth;
}
synchronized void northExit(int id) {
    --nNorth;
    if (nNorth == 0)
        notifyAll();
}
```

It's done in the same way for southbound cars but with corresponding symbols. Other than these operations we check which direction the car is going and whether it has crossed the bridge or not, because in that case we want to go back over the bridge.

On the next page is a sample of the output.

```
[49:217] Car 2> Waiting to go North
[49:219] Car 1> Going South
[49:219] Car 1> Crossing South with 3 trips left
[49:219] Car 3> Waiting to go North
[49:384] Car 1> Exiting South
[49:410] Car 1> Waiting to go North
[49:417] Car 0> Exiting South
[49:418] Car 2> Going North
[49:419] Car 2> Crossing North with 3 trips left
[49:419] Car 1> Going North
[49:421] Car 1> Crossing North with 3 trips left
[49:421] Car 3> Going North
[49:422] Car 3> Crossing North with 3 trips left
[49:457] Car 0> Going North
[49:458] Car 0> Crossing North with 3 trips left
[49:565] Car 1> Exiting North
[49:575] Car 2> Exiting North
```