APIT Lab Week5a

Description of solution

This is quite a tricky lab. The main idea is that the chef and the diner communicate via a shared buffer (the waiter). I.e. the chef can give dishes to the waiter when the waiter isn't currently holding a dish and the diner can take dishes from the waiter when they're ready.

All of the logic/locks/magic is going to be in the Waiter class.

The Chef class is very simple:

```
public class Chef extends Thread {
    private Waiter waiter;
    private String[] dishes = {"Starter","Main","Pudding","Coffee","Wafer thin mint"};
    public Chef(Waiter waiter) {
        this.waiter = waiter;
    }
    public void run() {
        for(String d: dishes) {
        try {
                Thread.sleep(3000); // time taken to prepare a course
            }catch(InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println("Chef has finished preparing " + d);
            waiter.giveDish(d);
        }
    }
}
```

It has a series of 5 dishes that can be prepared. It extends thread as it will run independently. On starting, it waits for three seconds (to simulate making the starter) and then tries to give the first dish (starter) to the waiter. Once it has managed this, it waits a further 3 seconds and tries to give the second dish to the waiter.

The diner is also pretty simple:

```
public class Diner extends Thread {
    private Waiter waiter;
    private boolean hasDish = false;
    private String dish;
    public Diner(Waiter waiter) {
        this.waiter = waiter;
    }
    public void run() {
        while(true) {
            this.dish = waiter.takeDish();
            System.out.println("Diner is consuming " + this.dish);
            try{
                Thread.sleep(5000); // time to eat it
            }catch(InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println("Diner has finished consuming " + this.dish);
        }
    }
}
```

It also extends Thread (as it will run independently) and, on starting, it attempts to take a dish from the waiter, once it has managed this it says it is consuming it and then sleeps for 5 seconds (the time it takes for this diner to eat). When the 5 seconds has elapsed it tries to take another dish, etc.

The waiter class is a bit more complicated:

```
import java.util.concurrent.locks.*;
public class Waiter {
    private ReentrantLock waiterLock = new ReentrantLock();
    private Condition waiterCondition = waiterLock.newCondition();
    private boolean hasDish = false;
    private String dish;
    public void giveDish(String dish) {
        // called by the chef
        waiterLock.lock();
        try {
            while(this.hasDish) {
                waiterCondition.await();
            }
            // when we get to here, the previous dish has gone so we can add a new one
            this.dish = dish;
            System.out.println("Chef has given the waiter the " + this.dish);
            this.hasDish = true;
            // Alert all the waiting things (the customer)
            waiterCondition.signalAll();

        }catch(InterruptedException e){
            e.printStackTrace();
        }finally {
            waiterLock.unlock();
        }
    }

    // Called by the customer
    public String takeDish() {
        waiterLock.lock();
        try{
            // Wait until there is a dish
            while(!this.hasDish) {
                waiterCondition.await();
            }
            // when we get here, there is a dish
            this.hasDish = false;
            System.out.println("Waiter has given " + this.dish + " to the diner");
            waiterCondition.signalAll();

        }catch(InterruptedException e) {
            e.printStackTrace();
        }finally{
            waiterLock.unlock();
        }
        return this.dish;
    }

}
```

It has a Reentrantlock and associated condition as well as a boolan field to store whether or not the waiter is currently holding a dish and a string to store the name of the dish.

Now consider `giveDish()`, this is called by `Chef`. It first locks the lock. Then it looks to see if the waiter is currently holding a dish. If the waiter is holding the dish, it calles `await` to wait until something else (it will be the diner) signals the condition. Once the condition is signaled, it checks again to see if the waiter has a dish. If it doesn't, it sets the hasDish to true (i.e. the waiter now has a dish) and sets the string dish to be the name. It then calls `signalAll` which tells all threads currently waiting that the status has changed. In other words, the Chef thread sits in the await until the waiter is not holding a dish, it then gives the waiter a dish and exits.

The `takeDish` method is similar. This is called by the Diner. It checks to see if there is a dish and waits if there isn't one. It uses await to wait and is therefore woken by a signalAll (invoked by the Chef thread once the waiter has been given a dish). Once a dish is present, the method sets `hasDish` to false, calls signalAll (to alert the potentially waiting Chef) and returns the name of the dish to the diner.

Finally, here's a main method that makes it all work.

```
public class Restaurant {
    public static void main(String[] args) {
        Waiter w = new Waiter();
        Chef c = new Chef(w);
        Diner d = new Diner(w);
        c.start();
        d.start();
    }
}
```