Implementation of dining philosopher in java

190031920

Nikhil Reddy Avuthu

Given problem statement

Five silent philosophers sit at a table around a bowl of spaghetti. A fork is placed between each pair of adjacent philosophers. (An alternative problem formulation uses rice and chopsticks instead of spaghetti and forks.)

Each philosopher must alternately think and eat. However, a philosopher can only eat spaghetti when he has both left and right forks. Each fork can be held by only one philosopher and so a philosopher can use the fork only if it's not being used by another philosopher. After he finishes eating, he needs to put down both forks so they become available to others. A philosopher can grab the fork on his right or the one on his left as they become available, but can't start eating before getting both of them.

Eating is not limited by the amount of spaghetti left assume an infinite supply. The problem is how to design a discipline of behavior (a concurrent algorithm) such that each philosopher won't starve, i.e., can forever continue to alternate between eating and thinking assuming that any philosopher cannot know when others may want to eat or think.

Documentation

Class: Dine - it has a main method.

Class: Philosopher - represent the philosopher, it contains the two objects to represent the chopsticks.

Methods

eat - called when enter for eating.

think - called when enter for thinking.

run - overridden method for Thread class.

Class: Chopstick - represent if chopstick is available.

Methods

take - called when before occupied.

release - called when philosopher starts thinking.

Both methods are the synchronized method.

To compile and run the program

javac Dine.java compiles the program

java Dine runs the program.

Dine.java

```
public class Dine {
  public static void main(String[] args) {


    int rounds = 10;


    Log.msg(String.valueOf(rounds));


    Chopstick[] chopistics = new Chopstick[5];


    for (int i = 0; i < chopistics.length; i++) {
      chopistics[i] = new Chopstick("C: " + i);
    }
```

```
        Philosopher[] philosophers = new Philosopher[5];

        philosophers[0] = new Philosopher("P: 0 - ", chopistics[0], chopistics[1], rounds);
        philosophers[1] = new Philosopher("P: 1 - ", chopistics[1], chopistics[2], rounds);
        philosophers[2] = new Philosopher("P: 2 - ", chopistics[2], chopistics[3], rounds);
        philosophers[3] = new Philosopher("P: 3 - ", chopistics[3], chopistics[4], rounds);
        philosophers[4] = new Philosopher("P: 4 - ", chopistics[0], chopistics[4], rounds);

        for (int i = 0; i < philosophers.length; i++) {
            Log.msg("Thread " + i + " has started");
            Thread t = new Thread(philosophers[i]);
            t.start();
        }
    }
}
```

Philosipher.java

```
public class Philosopher extends Thread {
    private Chopstick _leftChopistick;
    private Chopstick _rightChopistick;

    private String _name;
    private int _rounds;

    public Philosopher(String name, Chopstick _left, Chopstick _right, int rounds) {

        this._name = name;
        _leftChopistick = _left;
        _rightChopistick = _right;
        _rounds = rounds;
    }

    public void eat() {
        if (!_leftChopistick.used) {
            if (!_rightChopistick.used) {
                _leftChopistick.take();
                _rightChopistick.take();

                Log.msg(_name + " : Eat");

                Log.Delay(1000);

                _rightChopistick.release();
                _leftChopistick.release();
            }
        }
        think();
    }

    public void think() {

        Log.msg(_name + " : Think");
        Log.Delay(1000);

    }

    public void run() {
        for (int i = 0; i <= _rounds; i++) {
            eat();
        }
    }
}
```

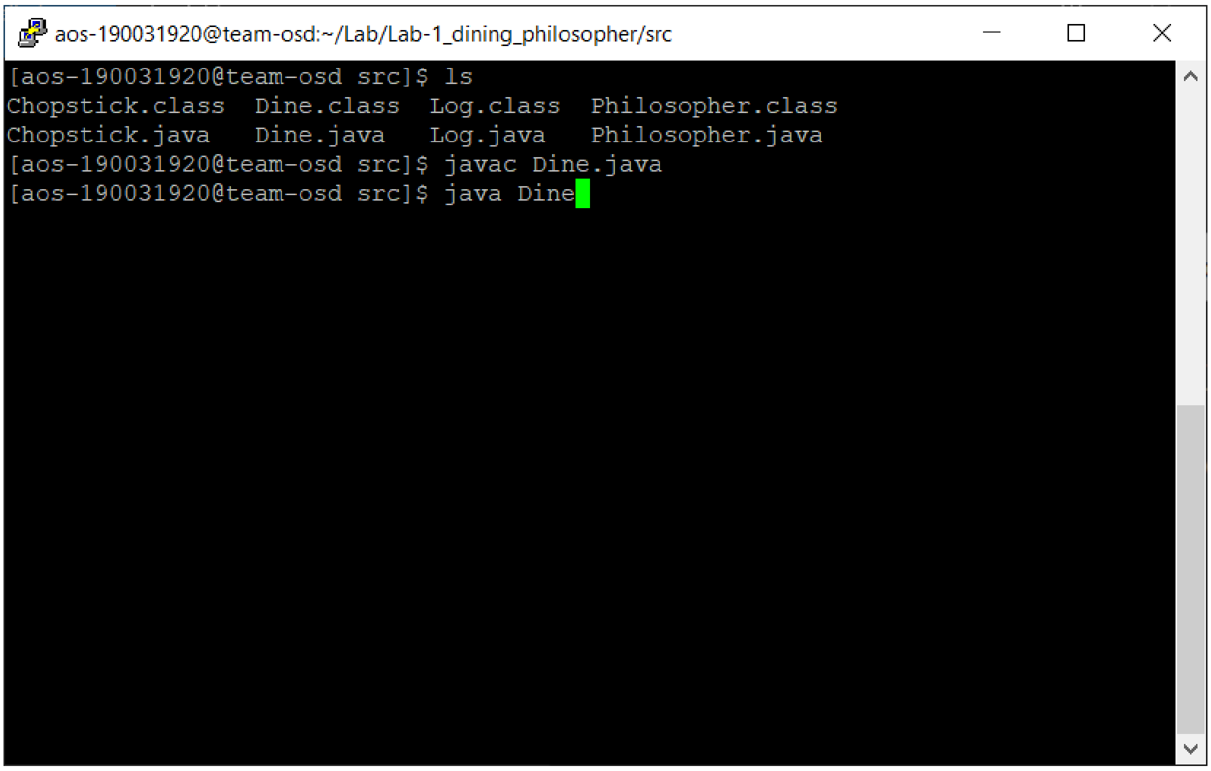
Log.java

```
public class Log {

    public static void msg(String msg) {
        System.out.println(msg);
    }

    public static void Delay(int ms) {
        try {
            Thread.sleep(ms);
        } catch (InterruptedException ex) {
        }
    }
}
```

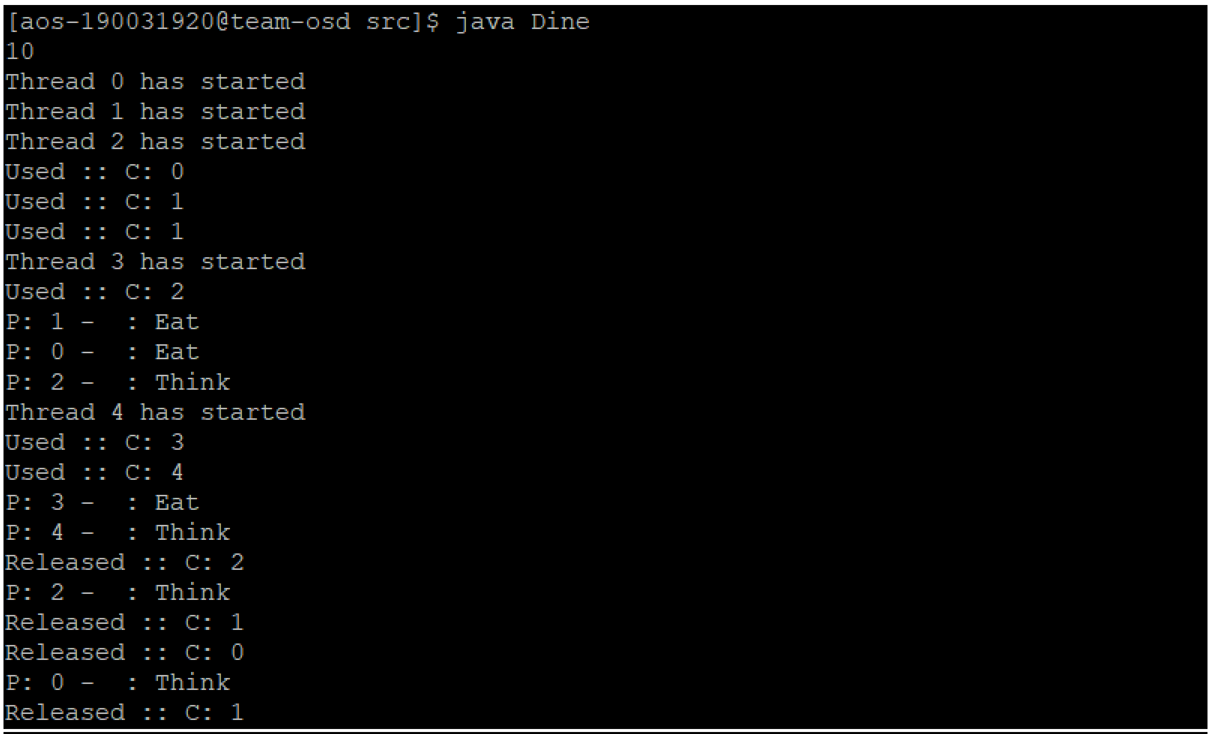
Chopstick.java

```
class Chopstick {
    public boolean used;
    public String _name;

    public Chopstick(String _name) {
        this._name = _name;
    }
    public synchronized void take() {
        Log.msg("Used :: " + _name);
        this.used = true;
    }

    public synchronized void release() {
        Log.msg("Released :: " + _name);
        this.used = false;
    }
}
```

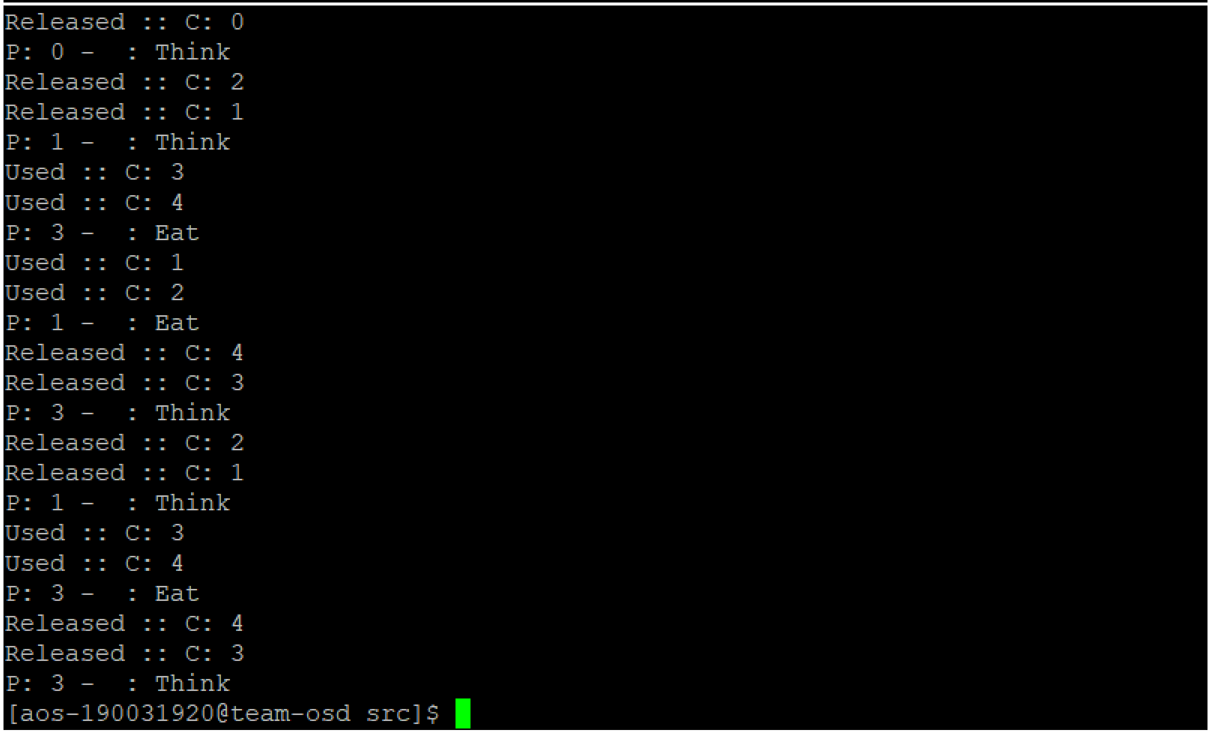
Output Screen shots:



```
[aos-190031920@team-osd src]$ ls
Chopstick.class  Dine.class  Log.class  Philosopher.class
Chopstick.java   Dine.java   Log.java   Philosopher.java
[aos-190031920@team-osd src]$ javac Dine.java
[aos-190031920@team-osd src]$ java Dine
```



```
[aos-190031920@team-osd src]$ java Dine
10
Thread 0 has started
Thread 1 has started
Thread 2 has started
Used :: C: 0
Used :: C: 1
Used :: C: 1
Thread 3 has started
Used :: C: 2
P: 1 -   : Eat
P: 0 -   : Eat
P: 2 -   : Think
Thread 4 has started
Used :: C: 3
Used :: C: 4
P: 3 -   : Eat
P: 4 -   : Think
Released :: C: 2
P: 2 -   : Think
Released :: C: 1
Released :: C: 0
P: 0 -   : Think
Released :: C: 1
```



```
Released :: C: 0
P: 0 -   : Think
Released :: C: 2
Released :: C: 1
P: 1 -   : Think
Used :: C: 3
Used :: C: 4
P: 3 -   : Eat
Used :: C: 1
Used :: C: 2
P: 1 -   : Eat
Released :: C: 4
Released :: C: 3
P: 3 -   : Think
Released :: C: 2
Released :: C: 1
P: 1 -   : Think
Used :: C: 3
Used :: C: 4
P: 3 -   : Eat
Released :: C: 4
Released :: C: 3
P: 3 -   : Think
[aos-190031920@team-osd src]$
```