一、源代码

```
1.  public class DiningPhilosophers {
2.      public static void main(String[] args) {
3.          int numPhilosophers = 5; // 哲学家数量
4.          Philosopher[] philosophers = new Philosopher[numPhilosophers];  //每个哲学家都是一个线程,执行吃饭等相关操作
5.          Chopstick[] chopsticks = new Chopstick[numPhilosophers];
6.
7.          //创建好桌上的筷子
8.          for (int i = 0; i < numPhilosophers; i++) {
9.              chopsticks[i] = new Chopstick();
10.         }
11.
12.         //哲学家入座，并且告诉他们对应的筷子是哪两支
13.         for (int i = 0; i < numPhilosophers; i++) {
14.             philosophers[i] = new Philosopher(i, chopsticks[i], chopsticks[(i + 1) % numPhilosophers]);
15.             philosophers[i].start();
16.         }
17.     }
18.
19.
20. }
21.
22. class Philosopher extends Thread {
23.     private int id;
24.     private boolean eaten;
25.     private Chopstick leftChopstick;
26.     private Chopstick rightChopstick;
27.
28.     public Philosopher(int id, Chopstick left, Chopstick right) {
29.         this.id = id;
30.         this.eaten = false;
31.         this.leftChopstick = left;
32.         this.rightChopstick = right;
33.     }
34.
35.     public void run() {
36.         while (!eaten) {
37.             think();
38.             pickUpChopsticks(); //此位置线程可能会进入 wait 阶段
39.             eat();
40.             putDownChopsticks();
41.         }
```

```
42.     }
43.
44.     /** 休眠随机时间 **/
45.     private void think() {
46.         System.out.println("Philosopher " + id + " is thinking");
47.         try {
48.             Thread.sleep((long) (Math.random() * 1000));
49.         } catch (InterruptedException e) {
50.             e.printStackTrace();
51.         }
52.     }
53.
54.     /** 此时应获取到筷子后，吃饭花费时间，不释放筷子的锁 **/
55.     private void eat() {
56.         System.out.println("Philosopher " + id + " is eating");
57.         try {
58.             Thread.sleep((long) (Math.random() * 1000));
59.             eaten = true;
60.             System.out.println("Philosopher " + id + " has finished eating");
61.         } catch (InterruptedException e) {
62.             e.printStackTrace();
63.         }
64.     }
65.
66.     /** 拿起筷子 即对两支筷子都加上同步锁 **/
67.     private void pickUpChopsticks() {
68.         synchronized (leftChopstick) {
69.             //左筷子不可用时则将筷子的锁释放
70.             while (!leftChopstick.isAvailable()) {
71.                 try {
72.                     //该线程将获取到的该对象的锁释放，并且等待被 notify
73.                     //即当前线程进入 wait 状态
74.                     leftChopstick.wait();
75.                 } catch (InterruptedException e) {
76.                     e.printStackTrace();
77.                 }
78.             }
79.             leftChopstick.pickUp();
80.         }
81.         synchronized (rightChopstick) {
82.             while (!rightChopstick.isAvailable()) {
83.                 try {
84.                     rightChopstick.wait();
85.                 } catch (InterruptedException e) {
```

```
                    e.printStackTrace();
                }
            }
            rightChopstick.pickUp();
        }
    }

    private void putDownChopsticks() {
        leftChopstick.putDown();
        rightChopstick.putDown();
        synchronized (leftChopstick) {
            //唤醒想用这根筷子的所有哲学家
            leftChopstick.notifyAll();
        }
        synchronized (rightChopstick) {
            rightChopstick.notifyAll();
        }
    }
}

class Chopstick {
    private boolean available = true;

    /**
     * synchronized 用于方法前表示仅有一个线程可以使用该方法
     **/
    public synchronized void pickUp() {
        available = false;
    }

    public synchronized void putDown() {
        available = true;
    }

    public synchronized boolean isAvailable() {
        return available;
    }
}
```

二、效果截图

```
Philosopher 0 is thinking
Philosopher 4 is thinking
Philosopher 3 is thinking
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 1 is eating
Philosopher 1 has finished eating
Philosopher 2 is eating
Philosopher 0 is eating
Philosopher 2 has finished eating
Philosopher 0 has finished eating
Philosopher 4 is eating
Philosopher 4 has finished eating
Philosopher 3 is eating
Philosopher 3 has finished eating
```

三、功能设计说明

此问题为经典的哲学家问题：假设有 5 位哲学家，他们坐在一个圆形桌子周围。每位哲学家面前有一碗米饭，但是只有 5 个筷子供他们使用，每位哲学家需要两只筷子才能进餐。每个哲学家思考和进餐的时间不确定，他们的行为由线程来模拟。

为了方便观察效果，我将此问题改为一位哲学家只进餐一次，进餐结束后不再进餐。虽然问题简化，但是还是会涉及到多位哲学家对一支筷子的同时竞争，因此需要设置同步锁。我使用的是 java 语言，就可以使用 synchronized 和 wait+notify 来实现进程间的通信。

每一位哲学家设为一个单独的线程，筷子作为全局数据被所有线程共享。当一位哲学家思考结束后，开始尝试获取筷子的使用权，先对左手筷子尝试获取同步锁，当左手筷子不可用的时候，释放筷子锁，进入线程等待状态，等待左筷子可用时被唤醒该线程；获取到左筷子后尝试获取右筷子，相同操作，当哲学家进餐完毕后，释放线程，并且激活筷子对象相关联的线程。

四、IPC 相关

在 java 中用 synchronized 获取对象或函数的锁，wait 或线程结束将会释放相关对象的锁，用 notify 能够唤醒对象相关联的线程，以此来进行线程通信，同时也需要在设计中避免死锁问题