

```

import java.util.concurrent.Semaphore;

public class Philosopher extends Thread{

    private int PhilosopherId;           //PhilosopherId
    static final int nrIterations =12;    //1000
    static final int nrTotalPhilosophers=7;
    static final int maxNrPhilosophersEating= (int)(nrTotalPhilosophers/2);
    static int nrPhilosophersEating =0;  // nr Philosophers eating (now)
    static int nrThreadsFinished =0;     // nr Philosophers finished (now)

    boolean eating = false; // true= yes; false= no (not eating) => thinks
    final boolean EATING = true; final boolean THINKING = false;

    final private static int minThinkTime =500; //ms
    final private static int thinkTime = 1200;  //ms
    final private static int minPauseMeal = 500; //ms
    final private static int varPauseMeal = 700; //ms
    private static Semaphore s = new Semaphore(maxNrPhilosophersEating);
    private static Semaphore mutex = new Semaphore(1); // to access shared var
nrPhilosophersEating
    private Philosopher leftPh, rightPh;
    private int nrMeals = 0;

    public Philosopher(int i ) {           // constructor
        PhilosopherId=i;                 setLeftPh(null);      setRightPh(null);}

                                                // getters and setters
    public static int getNrThreadsFinished() { return nrThreadsFinished;}
    public static void setNrThreadsFinished(int nrPhFinished) {
        nrThreadsFinished = nrPhFinished; }

    public int getNrMeals() { return nrMeals; }
    public void setNrMeals(int nrMeals) { this.nrMeals = nrMeals; }

    public Philosopher getLeftPh() { return leftPh; }
    public void setLeftPh(Philosopher leftPh) { this.leftPh = leftPh;}

    public Philosopher getRightPh() { return rightPh; }
    public void setRightPh(Philosopher rightPh) { this.rightPh = rightPh; }

    public static int getNrPhilosophersEating() { return nrPhilosophersEating; }
    public static void setNrPhilosophersEating(int nrPhilosophersEating) {
        Philosopher.nrPhilosophersEating = nrPhilosophersEating; }

    public int getPhilosopherId() { return PhilosopherId; }
    public void setPhilosopherId(int id) { this.PhilosopherId = id; }

    public boolean isEating() { return eating; }
    public void setEating(boolean eating) { this.eating = eating; }

    static int getLeftPhilosopherId(int i){
        return (int) ((i-1 + nrTotalPhilosophers) % nrTotalPhilosophers );}
    static int getRightPhilosopherId(int i){ return (int) ((i+1) % nrTotalPhilosophers );}

    private static void initData() {
        System.out.println ("***** Philosophers ***** \n");
        System.out.println ("Nr Philosophers = " + nrTotalPhilosophers + ".");
        System.out.println (" Max Nr Philosophers Eating (at same time) = " +
maxNrPhilosophersEating + ".\n");
    }

    private void philosopherThinks(int time, int iterations) {
        int thinks; thinks= (int)(minThinkTime+Math.random()*time);

        System.out.print ("\n Phil_id:" + getPhilosopherId() +" begins thinking "+ thinks+
" ms. NrPhilsEating:"+getNrPhilosophersEating()+".
Eats?:"+this.isEating()+
". Meals: *"+getNrMeals() +"* . Count: "+ iterations+ ".");

        try {Thread.sleep(thinks);} catch(InterruptedException e) {};
    }
}

```

```

private void philosopherEats(){
    int eatTime;

    eatTime= minPauseMeal+ (int)(Math.random()*varPauseMeal);
    setNrMeals(getNrMeals()+1);
    System.out.print ("\n Phil_id:" + getPhilosopherId() +" begins eating: "+eatTime+
        " ms. Meals: "+getNrMeals() +".
NrPhilsEating:"+getNrPhilosophersEating()+". Eats?:"+this.isEating()+".");

    try {Thread.sleep(eatTime);} catch(InterruptedException e) {};
}

private boolean triesToEat() {
    Philosopher l,r;
    boolean succeeds = true;

    try {mutex.acquire();} catch(InterruptedException e) {};

    l= getLeftPh();
    r= getRightPh();
    if((!l.isEating())&&(!r.isEating())) {
        try {s.acquire();} catch(InterruptedException e) {};
        this.setEating(EATING);
        setNrPhilosophersEating(getNrPhilosophersEating()+1);
        mutex.release();
        philosopherEats();
        try {mutex.acquire();} catch(InterruptedException e) {};
        this.setEating(THINKING);
        setNrPhilosophersEating(getNrPhilosophersEating()-1);
        mutex.release();
        s.release();
    }
    else {
        mutex.release();
        System.out.println ("\n Phil_id:" + getPhilosopherId() +
            " CAN'T EAT ... Meals: "+getNrMeals() +".
NrPhilsEating:"+getNrPhilosophersEating()+
            ". Eats? "+this.isEating()+". LeftPh :"+ getLeftPh().getPhilosopherId()+"
Eats? "+
            getLeftPh().isEating()
            +" RightPh :"+getRightPhilosopherId(getPhilosopherId())+" Eats?
"+getRightPh().isEating()+".");
        succeeds = false;
    }
    return succeeds;
}

private int antiStarvationCorrection(boolean eats) {
    return eats? 2*thinkTime:(int) (thinkTime/3); }

private synchronized void summary() {
    setNrThreadsFinished(getNrThreadsFinished()+1);
    System.out.println ("\n *** PhilosopherId:" + getPhilosopherId() +"
finished. Nr of meals: "+getNrMeals()
    +". Nr of Threads finished: "+ getNrThreadsFinished()+". ***\n");
}

public void run() {

    int moreChances;// anti Starvation correction
    boolean eats= false;

    for (int i=0;i<nrIterations;i++) { // while (true) {
        moreChances = antiStarvationCorrection(eats);

        philosopherThinks(moreChances,i);
        eats= triesToEat();
    }
    summary();
}

```

```

public static void main(String[] args) {
    initData();

    Philosopher [] p =new Philosopher [nrTotalPhilosophers];

    for(int i=0;i<nrTotalPhilosophers;i++) {    p[i]=new Philosopher (i);    }

    for(int i=0;i<nrTotalPhilosophers;i++) {
        p[i].setLeftPh(p[getLeftPhilosopherId(i)]);
        p[i].setRightPh(p[getRightPhilosopherId(i)]);
        p[i].start();
    }

    for(int i=0;i<nrTotalPhilosophers;i++) {
        System.out.println (" PhilosopherId:" + i + " LeftPhId:" +
getLeftPhilosopherId(i)
        + " RightPhId:"+getRightPhilosopherId(i)+".");
    }    // end for
    }//end main
}// end class

```