Solution in pseudocode :

```
class Chopstick:
   private boolean[] taking

   procedure Chopstick()
   {
      for i =0 To DiningTest.K :
         taking[i]=false
      Endfor
   }
   public synchronized procedure release(){

            Philosopher phi=(Philosopher) Thread.currentThread()
            Num=phi.Num
            display("Philosopher\t"+Num +"\treleases Chopstick\n")
            taking[Num]=false
taking[((Num+1) Mod DiningTest.K)]=false
notifyAll()
   }
   public synchronized procedure take(){
         Philosopher phi=(Philosopher) Thread.currentThread()
         Num=phi.Num

         while taking[((Num+1) Mod DiningTest.K)] Or taking[Num]
                  try :
                           wait()            catch
(InterruptedException e): Endwhile

         display ("Philosopher\t"+Num+"\ttakes Chopstick\n")
         taking[Num]=true          taking[((Num+1)
Mod DiningTest.K)]=true
   }
Endclass



class Philosopher extends Thread ():

Num
static Number=0
private Chopstick Chop
public procedure Philosopher(Chopstick Chop){
```

```
this.Chop=Chop
Num=Number
   Number=Number+1
   }

   private procedure hungry(){

      display ("Philosopher\t"+Num +"\tis Hungry\n")
   }

   private procedure eating(){

      display ("Philosopher\t"+Num +"\tis Eating\n")

try :

         Thread.sleep(500)

      catch (InterruptedException e) :
   }
   private procedure thinking(){

      display ("Philosopher\t"+Num +"\tis Thinking\n")

try :

         Thread.sleep(500)

      catch (InterruptedException e) :
   }

   public procedure run(){

      while(true){ thinking()
         hungry()
Chop.take()          eating()
         Chop.release()
      }
Endclass
```

```
public class DiningTest :

static K

   public static procedure main(){

      display("Enter number of Philosophers : ")

      n <- NUMBER(INPUT())

      K=n

      Chopstick CH = new Chopstick()

      for i =0 To K

         new Philosopher(CH).start()

      Endfor

   }

Endclass
```

Examples of Deadlock

```
do {
 wait (chopstick[i] ); // left chopstick
 wait (chopStick[ (i + 1) % DiningTest.K] ); // right chopstick
// eat
 signal (chopstick[i] );// left chopstick
 signal (chopstick[ (i + 1) % DiningTest.K] ); // right chopstick
 // think
 } while (TRUE);
```

Solving

we changed the (Wait – Signal) solution by (Monitor) solution with some edits as following :

in chopstick class :

```
public synchronized procedure release(){
        Philosopher phi=(Philosopher) Thread.currentThread()
        Num=phi.Num
        display("Philosopher\t"+Num +"\treleases Chopstick\n")
        taking[Num]=false  taking[((Num+1) Mod
        DiningTest.K)]=false  notifyAll()
}
public synchronized procedure take(){
        Philosopher phi=(Philosopher) Thread.currentThread()  Num=phi.Num
        while taking[((Num+1) Mod DiningTest.K)] Or taking[Num]
                try : wait()  catch
        (InterruptedException e):  Endwhile
        display ("Philosopher\t"+Num+"\ttakes Chopstick\n")  taking[Num]=true
         taking[((Num+1) Mod DiningTest.K)]=true}
```

Examples of Starvation

```
monitor DiningPhilosophers {   enum { THINKING, HUNGRY, EATING) state [5] ;
//Default for all 5 is THINKING
condition self [5];
void pickup (int i) {  state[i] = HUNGRY;  test(i);
                //try to acquire the two forks
        if (state[i] != EATING)  self[i].wait;   //block myself if the two
        forks isn't acquired
         }
void putdown (int i) {
  state[i] = THINKING;  //philosopher has finished eating  test((i +
        DiningTest.K-1) % DiningTest.K);  //see if left can now eat
   test((i + 1) % DiningTest.K);            //see if right can now eat
         }
```

Solving

we made the Philosopher we hungry first will eat first as following :

in Philosopher class :

```
private procedure hungry(){ display ("Philosopher\t"+Num
        +"\tis Hungry\n")
} private procedure eating(){ display ("Philosopher\t"+Num
+"\tis Eating\n") try :
                Thread.sleep(500) catch
        (InterruptedException e) :
 }
 private procedure thinking(){
            display ("Philosopher\t"+Num +"\tis Thinking\n")
        try :
                Thread.sleep(500)    catch
(InterruptedException e) :
 } public procedure
run(){ while(true){
thinking() hungry()
Chop.take() eating()
        Chop.release()
 }
```

Real world application:

We used a hotel reservation system as a Real world application for dining philosophers problem solving and we considered the philosophers as a customers who wants to book a room and we considered the room as the chopsticks so if number of customers is K so number of rooms is K/2 as only K/2 customers who can join the room like the problem only K/2 philosophers who can eat.

The pseudocode :
Room class is similar to Chopstick class :

```
class Room:
   private boolean[] taking

   procedure Room()
   {
      for i =0 To real_World_Application.K :
         taking[i]=false
      Endfor
   }
   public synchronized procedure leave (){

            Customer cust=(Customer) Thread.currentThread()
            Num=cust.Num
               display("Customer\t"+Num +"\tleaves the room \n")
     taking[Num]=false            taking[((Num+1) Mod
real_World_Application.K)]=false        notifyAll()
   }
   public synchronized procedure book(){
          Customer cust=(Customer) Thread.currentThread()
        Num=cust.Num

    while
taking[((Num+1) Mod real_World_Application.K)] Or
taking[Num]
                  try :
                  wait()             catch
(InterruptedException e): Endwhile

         display ("Customer\t"+Num+"\tbook  a room\n")
taking[Num]=true
taking[((Num+1) Mod real_World_Application.K)]=true
   }
Endclass
```

The Customer class is similar to philosopher class :

```
class Customer extends Thread ():
Num      static Number=0  private
Chopstick Chop     public
procedure Customer(Room room){
   this.room=room
Num=Number
   Number=Number+1
```

```
    }
    private procedure in (){

        display ("Customer\t"+Num +"\tis in the room\n")

try :

            Thread.sleep(500)

        catch (InterruptedException e) :
    }
    private procedure out(){

        display (“Customer\t"+Num +"\tis out of the room\n")

try :

            Thread.sleep(500)

        catch (InterruptedException e) :
    }

    public procedure run(){


while(true){
out()
room.book()
in()
room.leave()
        }
Endclass
```