i219 Software Design Methodology
12. Case study 1
Dining philosopher problem

Kazuhiro Ogata (JAIST)

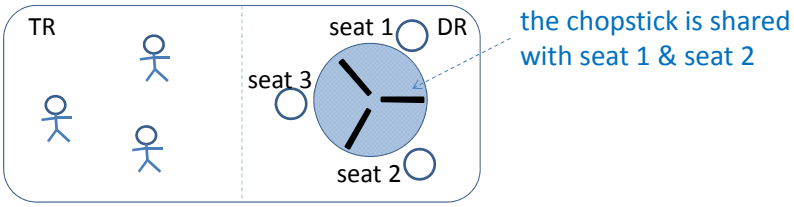
Outline of lecture

- Dining philosopher problem (DPP)
- Dining Room in UML & Java
- Chopstick in UML & Java
- Philosopher in UML & Java
- DPP in UML & Java
- Analysis of DPP

Dining philosopher problem (DPP) (1)

- There are n ($\geqq 2$) philosophers who are either thinking in a thinking room TR or eating in a dining room DR.
- There is a table in DR that has n seats.

  Each seat is given a pair of chopsticks *left* & *right*, but they are supposed to be shared with the philosophers at the left & right seats, respectively.

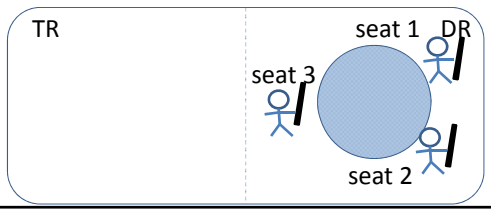


Dining philosopher problem (DPP) (2)

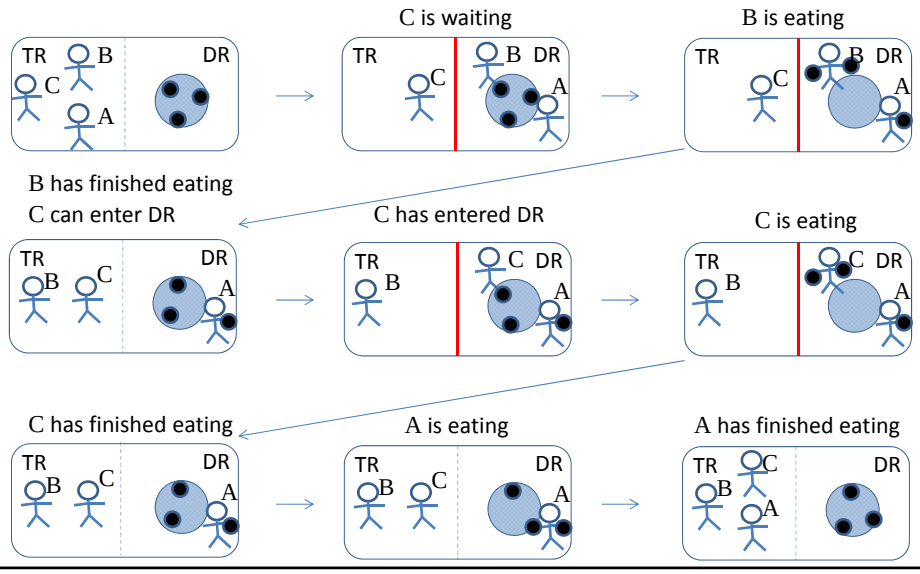
- p can eat only if p holds both *left* & *right*.

  Once p holds *left*, *right* or both of them, p never releases them until p finishes eating.
- Possible to prevent deadlock?

  If every philosopher holds one chopstick, it is deadlock.
- One solution is to allow at most $n-1$ philosophers to enter DR.



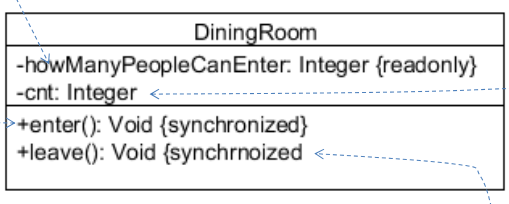
Dining philosopher problem (DPP) (3)



Dining Room in UML & Java (1)

DR has the capacity (how many people can enter) that is represented by

cnt is the number of people in DR



A person can enter DR if cnt is less than the capacity, and increments cnt; otherwise, he/she needs to wait until at least another person leaves DR

A person in DR can leave DR, decrements cnt, and let waiting people know that one seat becomes available

Dining Room in UML & Java (2)

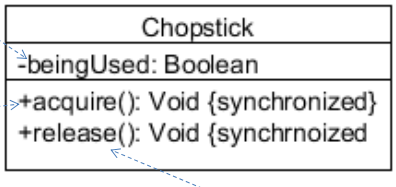
```
public class DiningRoom {
  private final int howManyPeopleCanEnter;  private int cnt;
  public DiningRoom(int howMany) {
    this.howManyPeopleCanEnter = howMany;  this.cnt = 0;  }
  public synchronized void enter() throws InterruptedException {
    if (howManyPeopleCanEnter > cnt) { cnt++; }
    else {
      while (howManyPeopleCanEnter <= cnt) { this.wait(); }
      cnt++;
    }  }
  public synchronized void leave() {
    cnt--;
    this.notifyAll();
  }
}
```

a philosopher may have to wait until at least one other philosopher leaves DR

every time a philosopher leaves DR, he/she lets other philosophers know it

Chopstick in UML & Java (1)

beingUsed is true if the chopstick is used and false otherwise



A person can use the chopstick if beingUsed is false, and sets it to true; otherwise, he/she needs to wait until the chopstick is released

A person can release the chopstick used by him/her, sets beingUsed to false, and lets people waiting for the chopstick know that the chopstick becomes available

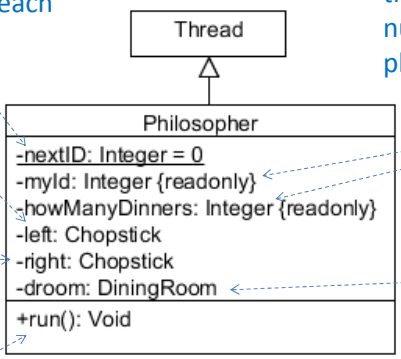
Chopstick in UML & Java (2)

```
public class Chopstick {
  private boolean beingUsed;
  public Chopstick() { this.beingUsed = false; }
  public synchronized void acquire() throws InterruptedException {
    while (beingUsed) { this.wait(); }
    beingUsed = true;
  }
  public synchronized void release() {
    beingUsed = false;
    this.notifyAll();
  }
}
```

a philosopher has to wait while the chopstick is being used

every time a philosopher releases the chopstick, he/she lets other philosophers know it

Philosopher in UML & Java (1)



nextId is used to give each philosopher a unique identification number

the identification number given to the philosopher

the left chopstick shared by the left philosopher

how many dinners the philosopher will take

the right chopstick shared by the right philosopher

the dining room

each philosopher conducts the six actions for each dinner, repeating them howManyDinners times

entering DR
acquiring the left chopstick
acquiring the right chopstick
releasing the right chopstick
releasing the left chopstick
leaving DR

Philosopher in UML & Java (2)

```
public class Philosopher extends Thread {
  private static int nextId = 0;              private final int myId;
  private final int howManyDinners;    private Chopstick left;
  private Chopstick right;                    private DiningRoom droom;

  public Philosopher(int n,Chopstick l,Chopstick r,DiningRoom dr) {
    this.myId = nextId++;          this.howManyDinners = n;
    this.left = l;    this.right = r;    this.droom = dr;
    }

    public void run() { … }
}
```

a unique identification number is given to the philosopher object being created

Philosopher in UML & Java (3)

```
public void run() {
  for (int i = 0; i < howManyDinners; i++) {
    // thinking
    try { droom.enter(); }
    catch (InterruptedException e) {}
    try { left.acquire(); }
    catch (InterruptedException e) {}
    try { right.acquire(); }
    catch (InterruptedException e) {}
    // taking a dinner
    right.release();
    left.release();
    droom.leave();
}
```

entering DR

acquiring the left chopstick

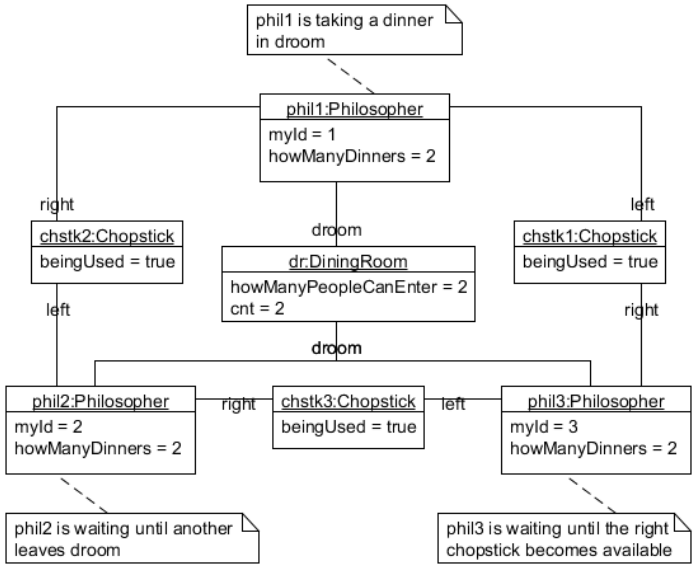
acquiring the right chopstick

releasing the right chopstick

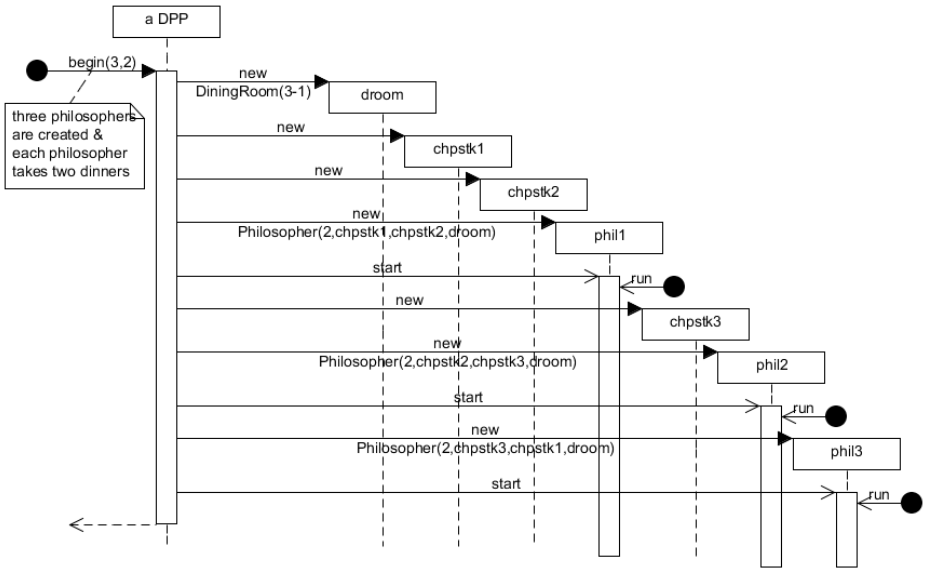
releasing the left chopstick

leaving DR

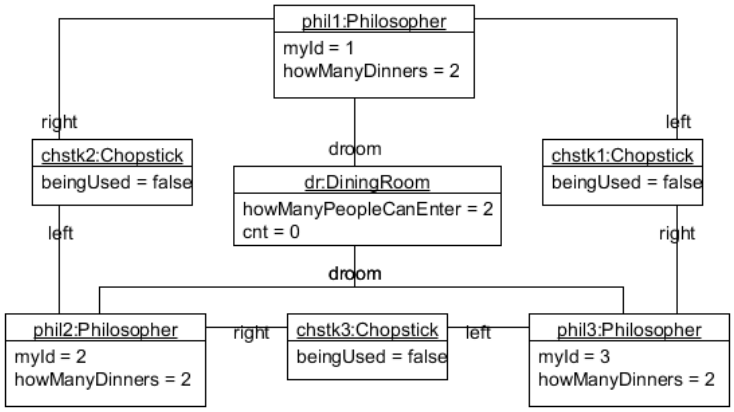
DPP in UML & Java (1)



DPP in UML & Java (2)



DPP in UML & Java (3)



DPP in UML & Java (4)

```
public class DiningPhilosopherProblem {
  public void begin(int n,int m) {
    DiningRoom dr = new DiningRoom(n-1);
    Chopstick left;
    Chopstick right = new Chopstick();
    Chopstick tmp = right;
    for (int i = 0; i < n-1; i++) {
      left = right;
      right = new Chopstick();
      (new Philosopher(m,left,right,dr)).start();
    }
    left = right;
    right = tmp;
    (new Philosopher(m,left,right,dr)).start();
  }
}
```

n philosophers; each eats m times

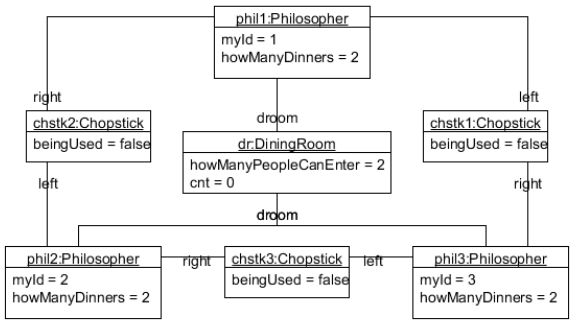
DR is created; at most n–1 philosophers are allowed to enter DR at the same time

i^{th} philosopher thread is created for $i = 1 \ldots n - 1$; they are scheduled by sending start() to them

n^{th} philosopher thread is created & scheduled by sending start() to it

DPP in UML & Java (5)

```
public class Test3DPP {
   public static void main(String[] args) {
      DiningPhilosopherProblem dpp
         = new DiningPhilosopherProblem();
      dpp.begin(3,2);
   }
}
```



Analysis of DPP (1)

In Test3DPP.jpf:

```
target = Test3DPP
classpath+=.
sourcepath+=.
report.console.property_violation=error,trace,snapshot
```

Analysis of DPP (2)

JPF has reported that there is no error detected.

```
====================================================== search started: 17/01/10 11:18

====================================================== results
no errors detected

====================================================== statistics
elapsed time:       00:00:49
states:             new=286086,visited=608580,backtracked=894666,end=226
search:             maxDepth=167,constraints=0
choice generators:  thread=286086
(signal=17865,lock=75500,sharedRef=145954,threadApi=3,reschedule=46764), data=0
heap:               new=24031,released=971448,maxLive=381,gcCycles=773501
instructions:       5016534
max memory:         417MB
loaded code:        classes=66,methods=1484

====================================================== search finished: 17/01/10 11:19
```

on Windows 7 with Intel® Core™ i7-2620M CPU @ 2.70GHz and 8GB memory

Analysis of DPP (3)

Let us model check DPP in which there are five philosophers and each philosopher takes one dinner.

```
public class Test5DPP {
   public static void main(String[] args) {
      DiningPhilosopherProblem dpp
         = new DiningPhilosopherProblem();
      dpp.begin(5,1);
   }
}
```

This model checking requires more than the default memory size (1024 MB) used by JPF. So, the memory size used should be specified as follows:

```
% java -Xmx7168m -ea
   -jar /Users/ogata/projects/jpf-core/build/RunJPF.jar
   Test5DPP.jpf
```

Analysis of DPP (4)

JPF has reported that there is no error detected.

```
====================================================== system under test
Test5DPP.main()

====================================================== search started: 17/01/01 16:22

====================================================== results
no errors detected

====================================================== statistics
elapsed time:       03:02:10
states:             new=37203228,visited=141925031,backtracked=179128259,end=191
search:             maxDepth=144,constraints=0
choice generators:  thread=37203228
(signal=1955720,lock=10008827,sharedRef=17684099,threadApi=5,reschedule=7554577),
data=0
heap:               new=5665551,released=157861614,maxLive=391,gcCycles=148046996
instructions:       1004192967
max memory:         1226MB
loaded code:        classes=66,methods=1484

====================================================== search finished: 17/01/01 19:24
```

on Windows 7 with Intel® Core™ i7-2620M CPU @ 2.70GHz
and 8GB memory

Analysis of DPP (5)

Let us model check DPP in which there are five philosophers and each philosopher takes two dinners.

```
public class Test5DPP {
   public static void main(String[] args) {
      DiningPhilosopherProblem dpp
         = new DiningPhilosopherProblem();
      dpp.begin(5,2);
   }
}
```

The following amount of memory size was used:

```
% java –Xmx30720m -ea
   -jar /Users/ogata/projects/jpf-core/build/RunJPF.jar
   Test5DPP.jpf
```

Analysis of DPP (6)

The model checking could not be conducted because of out of memory:

```
====================================================== search started: 17/01/06 7:18
[SEVERE] JPF out of memory
…
====================================================== results
error #1: gov.nasa.jpf.vm.NoOutOfMemoryErrorProperty

====================================================== statistics
elapsed time:       41:11:00
states:             new=751599616,visited=2849662858,backtracked=3601262363,end=373
search:             maxDepth=285,constraints=1
choice generators:  thread=751599616
(signal=48396150,lock=221054680,sharedRef=360687123,threadApi=5,reschedule=121461658)
data=0
heap:               new=408,released=2463530368,maxLive=391,gcCycles=-1382241218
instructions:       16898260305
max memory:         17860MB
loaded code:        classes=66,methods=1484

====================================================== search finished: 17/01/08 0:29
```

on Mac OS X 10.9.5 with 3.4 GHz Intel® Core™ i7Intel®
and 32 GB memory

Summary

- Dining philosopher problem (DPP)
- Dining Room in UML & Java
- Chopstick in UML & Java
- Philosopher in UML & Java
- DPP in UML & Java
- Analysis of DPP