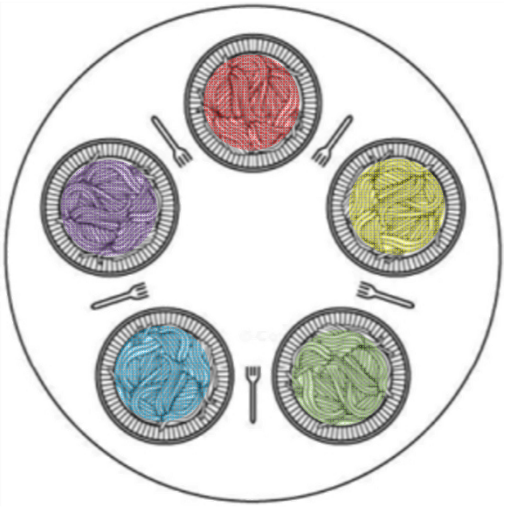
CSE216 Programming Abstractions
Deadlock

YoungMin Kwon

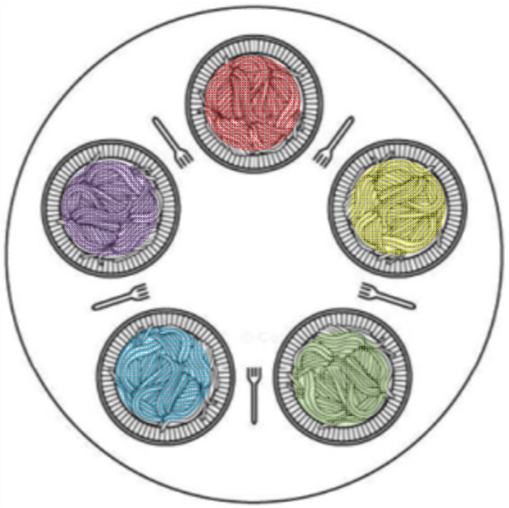
Dining Philosophers Problem

- 5 Philosophers are
  - Thinking
  - Grabbing two forks
    - Left-side then right-side
  - Eating
  - Putting the forks down
  - Repeat…



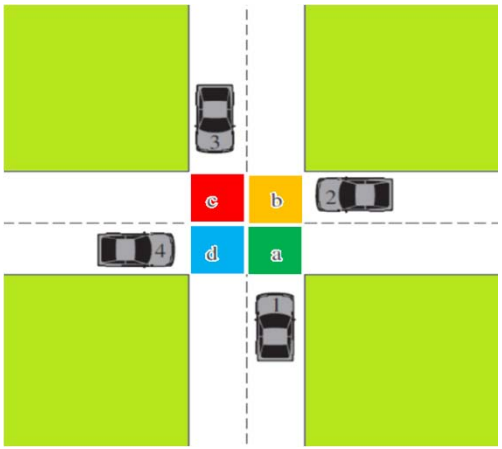
Dining Philosophers Problem

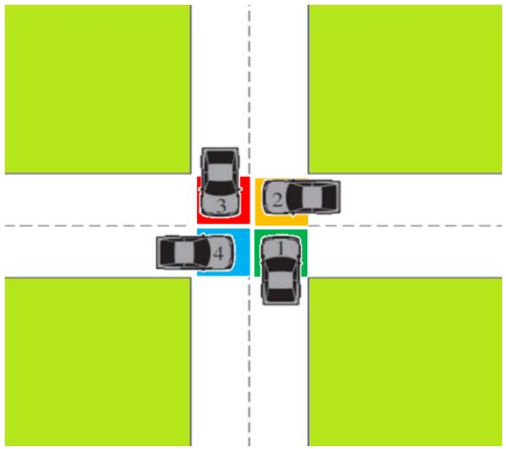
- What happens if
  - All philosophers pick their left-side forks together?
  - While waiting for their right-side forks, they will starve to die



Cars Entering an Intersection

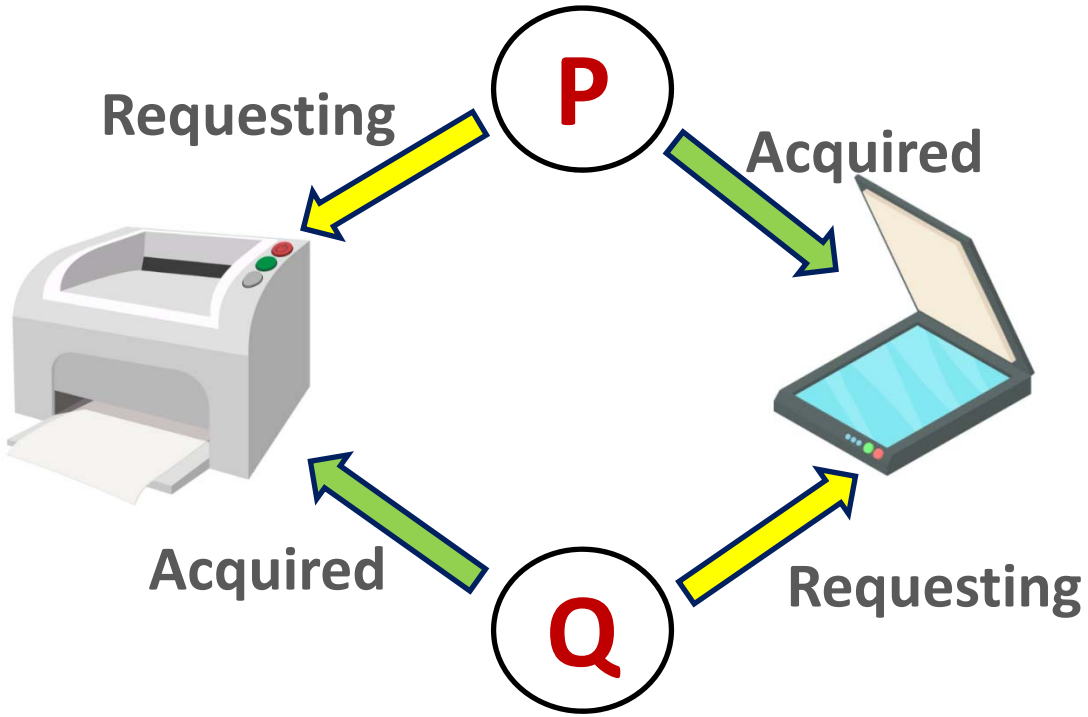
- To pass an intersection, a car has to pass through two small square blocks
- What happens if all 4 cars enter the intersection together?





Processes Requesting Resources

- Two processes trying to use a printer and a scanner

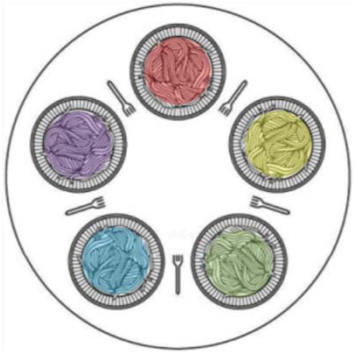


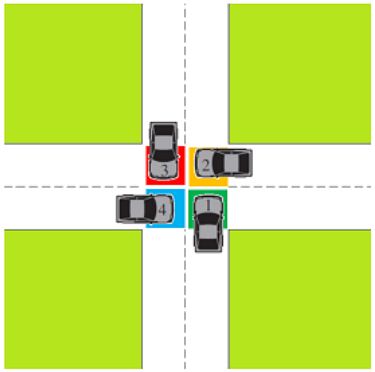
Deadlock

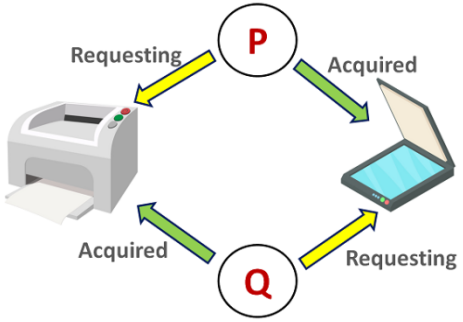
- A set of processes are deadlocked if
  - Each process in the set is blocked and
  - Waiting for an event that can be triggered only from another process in the set

Deadlock Conditions

- 4 deadlock conditions
  - Mutual exclusion: each resource is assigned to exactly one thread
  - Hold and wait: threads currently holding resources can request new resources
  - No preemption: resources previously granted cannot be forcefully taken away
  - Circular wait: circular chain of two or more threads waiting for the resources held by others







Dining Philosophers Problem in Java

```
public class DiningPhilosophers {
    public static class Fork {  //forks are shared resources
        ReentrantLock lock;     //mutual exclusion to use the fork
        String name;

        public Fork(String name) {
            this.name = name;
            lock = new ReentrantLock();
        }

        public void get() {
            System.out.format("%s: getting...\n", name);
            lock.lock();    //acquire the lock to use it
            System.out.format("%s: getting done\n", name);
        }

        public void put() {
            System.out.format("%s: putting...\n", name);
            lock.unlock();  //release the lock when finished using it
            System.out.format("%s: putting done\n", name);
        }
    }
```

```
public static class Philosopher implements Runnable {
    Fork l, r;
    String name;
    public Philosopher(String name, Fork l, Fork r) {
        this.l = l; this.r = r;
        this.name = name;
    }

    public void run() {
        for(int i = 0; i < 100; i++) {
            //thinking
            System.out.format("%s: thinking...\n", name);
            for(int j = 0; j < 1000000; j++) /*thinking*/;
            System.out.format("%s: thinking done\n", name);

            //getting forks
            System.out.format("%s: getting left...\n", name);
            l.get();
            System.out.format("%s: getting left done\n", name);

            System.out.format("%s: getting right...\n", name);
            r.get();
            System.out.format("%s: getting right done\n", name);
```

…

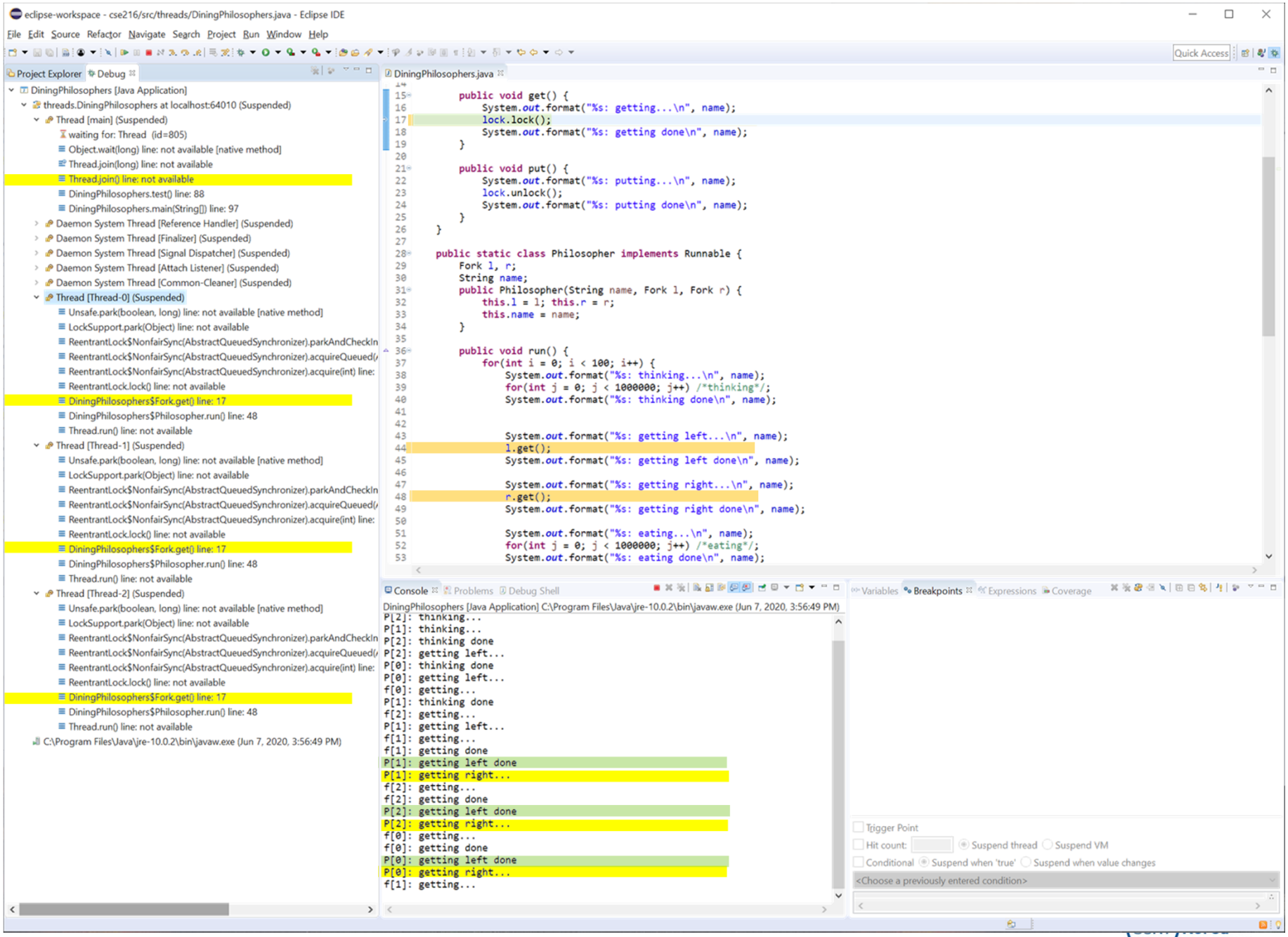
…

```
                //eating
                System.out.format("%s: eating...\n", name);
                for(int j = 0; j < 1000000; j++) /*eating*/;
                System.out.format("%s: eating done\n", name);

                //putting forks
                System.out.format("%s: puttting left...\n", name);
                l.put();
                System.out.format("%s: putting left done\n", name);

                System.out.format("%s: putting right...\n", name);
                r.put();
                System.out.format("%s: putting right done\n", name);
            }
        }

        //Factory class
        public static class Factory {
            public Philosopher create(String name, Fork l, Fork r) {
                return new Philosopher(name, l, r);
            }
        }
    }
```

```
public static void test(Philosopher.Factory factory) {
    int n = 3;
    try {
        //create Forks
        Fork[] forks = new Fork[n];
        for(int i = 0; i < n; i++)
            forks[i] = new Fork(String.format("f[%d]", i));

        //create Philosophers
        Philosopher[] phil = new Philosopher[n];
        for(int i = 0; i < n; i++)
            phil[i] = factory.create(String.format("P[%d]", i),
                                     forks[i], forks[(i+1) % n]);

        //run Philosophers on their own threads
        Thread[] threads = new Thread[n];
        for(int i = 0; i < n; i++) {
            threads[i] = new Thread(phil[i]);
            threads[i].start();
        }

        //wait for the Philosophers to finish
        for(int i = 0; i < n; i++)
            threads[i].join();
        System.out.println("Done!");
    } catch(Exception e) { e.printStackTrace(); }
}
```

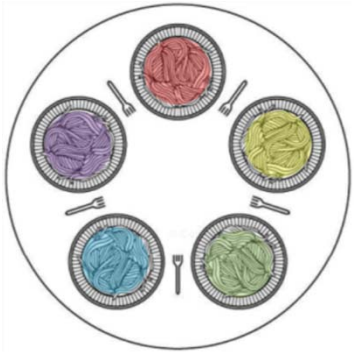
```
    public static void main(String[] args) {
        test(new Philosopher.Factory());
    }
}
```

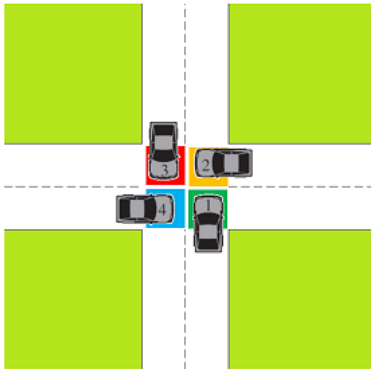
```
Output:
P[0]: thinking...
P[2]: thinking...
P[1]: thinking...
P[2]: thinking done
P[2]: getting left...
P[0]: thinking done
P[0]: getting left...
f[0]: getting...
P[1]: thinking done
f[2]: getting...
P[1]: getting left...
f[1]: getting...
f[1]: getting done
P[1]: getting left done
P[1]: getting right...
f[2]: getting...
f[2]: getting done
P[2]: getting left done
P[2]: getting right...
f[0]: getting...
f[0]: getting done
P[0]: getting left done
P[0]: getting right...
f[1]: getting...
```

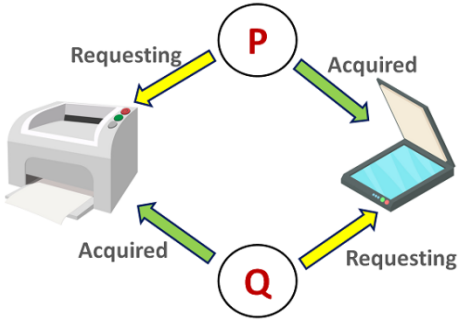

eclipse-workspace - cse216/src/threads/DiningPhilosophers.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Project Explorer
Debug
DiningPhilosophers [Java Application]
threads.DiningPhilosophers at localhost:64010 (Suspended)
Thread [main] (Suspended)
waiting for: Thread (id=805)
Object.wait(long) line: not available [native method]
Thread.join(long) line: not available
Thread.join() line: not available
DiningPhilosophers.test() line: 88
DiningPhilosophers.main(String[]) line: 97
Daemon System Thread [Reference Handler] (Suspended)
Daemon System Thread [Finalizer] (Suspended)
Daemon System Thread [Signal Dispatcher] (Suspended)
Daemon System Thread [Attach Listener] (Suspended)
Daemon System Thread [Common-Cleaner] (Suspended)
Thread [Thread-0] (Suspended)
Unsafe.park(boolean, long) line: not available [native method]
LockSupport.park(Object) line: not available
ReentrantLock$NonfairSync(AbstractQueuedSynchronizer).parkAndCheckIn
ReentrantLock$NonfairSync(AbstractQueuedSynchronizer).acquireQueued(
ReentrantLock$NonfairSync(AbstractQueuedSynchronizer).acquire(int) line:
ReentrantLock.lock() line: not available
DiningPhilosophers$Fork.get() line: 17
DiningPhilosophers$Philosopher.run() line: 48
Thread.run() line: not available
Thread [Thread-1] (Suspended)
Unsafe.park(boolean, long) line: not available [native method]
LockSupport.park(Object) line: not available
ReentrantLock$NonfairSync(AbstractQueuedSynchronizer).parkAndCheckIn
ReentrantLock$NonfairSync(AbstractQueuedSynchronizer).acquireQueued(
ReentrantLock$NonfairSync(AbstractQueuedSynchronizer).acquire(int) line:
ReentrantLock.lock() line: not available
DiningPhilosophers$Fork.get() line: 17
DiningPhilosophers$Philosopher.run() line: 48
Thread.run() line: not available
Thread [Thread-2] (Suspended)
Unsafe.park(boolean, long) line: not available [native method]
LockSupport.park(Object) line: not available
ReentrantLock$NonfairSync(AbstractQueuedSynchronizer).parkAndCheckIn
ReentrantLock$NonfairSync(AbstractQueuedSynchronizer).acquireQueued(
ReentrantLock$NonfairSync(AbstractQueuedSynchronizer).acquire(int) line:
ReentrantLock.lock() line: not available
DiningPhilosophers$Fork.get() line: 17
DiningPhilosophers$Philosopher.run() line: 48
Thread.run() line: not available
C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (Jun 7, 2020, 3:56:49 PM)
DiningPhilosophers.java
15 public void get() {
16 System.out.format("%s: getting...\n", name);
17 lock.lock();
18 System.out.format("%s: getting done\n", name);
19 }
20
21 public void put() {
22 System.out.format("%s: putting...\n", name);
23 lock.unlock();
24 System.out.format("%s: putting done\n", name);
25 }
26 }
27
28 public static class Philosopher implements Runnable {
29 Fork l, r;
30 String name;
31 public Philosopher(String name, Fork l, Fork r) {
32 this.l = l; this.r = r;
33 this.name = name;
34 }
35
36 public void run() {
37 for(int i = 0; i < 100; i++) {
38 System.out.format("%s: thinking...\n", name);
39 for(int j = 0; j < 1000000; j++) /*thinking*/;
40 System.out.format("%s: thinking done\n", name);
41
42
43 System.out.format("%s: getting left...\n", name);
44 l.get();
45 System.out.format("%s: getting left done\n", name);
46
47 System.out.format("%s: getting right...\n", name);
48 r.get();
49 System.out.format("%s: getting right done\n", name);
50
51 System.out.format("%s: eating...\n", name);
52 for(int j = 0; j < 1000000; j++) /*eating*/;
53 System.out.format("%s: eating done\n", name);
Console
Problems
Debug Shell
DiningPhilosophers [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (Jun 7, 2020, 3:56:49 PM)
P[2]: thinking...
P[1]: thinking...
P[2]: thinking done
P[2]: getting left...
P[0]: thinking done
P[0]: getting left...
f[0]: getting...
P[1]: thinking done
f[2]: getting...
P[1]: getting left...
f[1]: getting...
f[1]: getting done
P[1]: getting left done
P[1]: getting right...
f[2]: getting...
f[2]: getting done
P[2]: getting left done
P[2]: getting right...
f[0]: getting...
f[0]: getting done
P[0]: getting left done
P[0]: getting right...
f[1]: getting...
Variables
Breakpoints
Expressions
Coverage
Trigger Point
Hit count:
Suspend thread
Suspend VM
Conditional
Suspend when 'true'
Suspend when value changes
<Choose a previously entered condition>

Deadlock Prevention

- Break one of the four deadlock conditions
  - Mutual exclusion: not always possible
  - No preemption: not always possible
  - Hold and wait: if a request cannot be granted immediately, release all acquired resources
  - Circular wait: give an order to resources and acquire them in the order







Dining Philosophers Problem

- Fixing the deadlock
  - By breaking the circular wait
  - Give orders to the resources
    - Lexicographical ordering on the name of the forks
  - SmartPhilosophers acquire forks in the name order

```
public static class SmartPhilosopher extends Philosopher {
    //Constructor
    public SmartPhilosopher(String name, Fork l, Fork r) {
        super(name, l, r);
    }

    //override Runnable
    public void run() {
        for(int i = 0; i < 100; i++) {
            System.out.format("%s: thinking...\n", name);
            for(int j = 0; j < 1000000; j++) /*thinking*/;
            System.out.format("%s: thinking done\n", name);

            //getting forks based on their names
            Fork f = l, s = r;
            if(r.name.compareTo(l.name) < 0) {
                f = r; s = l;
            }
            System.out.format("%s: getting first...\n", name);
            f.get();
            System.out.format("%s: getting first done\n", name);

            System.out.format("%s: getting second...\n", name);
            s.get();
            System.out.format("%s: getting second done\n", name);
```

```
            System.out.format("%s: eating...\n", name);
            for(int j = 0; j < 1000000; j++) /*eating*/;
            System.out.format("%s: eating done\n", name);

            //putting forks
            System.out.format("%s: puttting left...\n", name);
            l.put();
            System.out.format("%s: putting left done\n", name);

            System.out.format("%s: putting right...\n", name);
            r.put();
            System.out.format("%s: putting right done\n", name);
        }
    }

    //Factory class
    public static class Factory extends Philosopher.Factory {
        public Philosopher create(String name, Fork l, Fork r) {
            return new SmartPhilosopher(name, l, r);
        }
    }
}
```

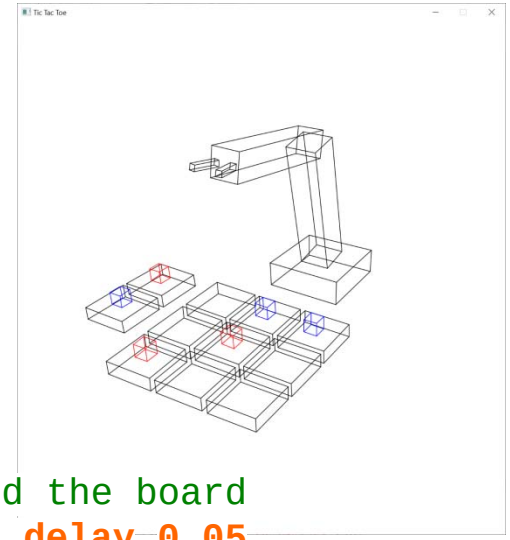
```
 public static void main(String[] args) {
        //test(new Philosopher.Factory());
        test(new SmartPhilosopher.Factory());
    }
}
```

Output

```
…
P[2]: getting first...
f[0]: getting...
f[0]: getting done
P[2]: getting first done
P[2]: getting second...
f[2]: getting...
f[2]: getting done
P[2]: getting second done
P[2]: eating...
P[2]: eating done
P[2]: puttting left...
f[2]: putting...
f[2]: putting done
P[2]: putting left done
P[2]: putting right...
f[0]: putting...
f[0]: putting done
P[2]: putting right done
Done!
```

Multi-threaded Application Design Example: Tic-Tac-Toe Robot

- Two issues to fix in the previous implementations
  - Upper layer functions have to wait for the animation delays

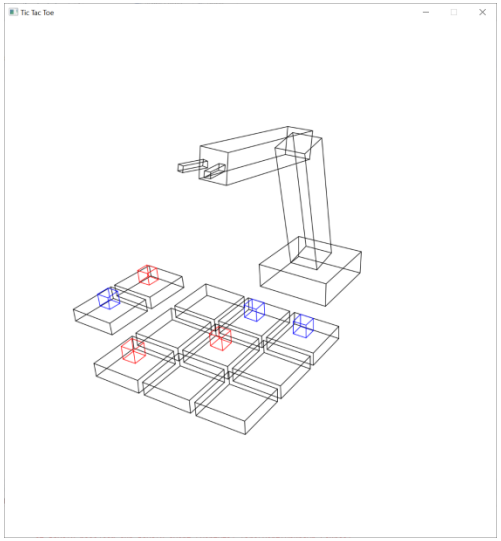


```
let rot_joint pose joint ang step =
        (*TODO: implement this method
            - on each step, draw the robot and the board
            - wait for 50ms by calling Thread.delay 0.05
            - then either return pose or rotate more
```

  - Difficult to plug-in new components
    - E.g. adding a remote player over a network

Multi-threaded Application Design

- Threaded components
  - EventScheduler
  - Game
  - Player
    - PlayerComputer
    - PlayerHuman



Event Scheduler

- Queues events with delays
- Fires events at a specified moment

- Example
  - A sequence of Joint rotations with delays can be scheduled at the event scheduler
  - The drawing is performed on the Event scheduler's thread
  - The other parts of the code do not need to wait for the delayed operations to finish

```
public class Event {
    public long fireAt;  //msec
    Function<Event,Integer> handler; //callback

    public Event(Function<Event,Integer> handler) {
        this.handler = handler;
        fireAt = 0;
    }

    //register handler to the global scheduler
    public void fireAfter(int msec/*delay*/) {
        Globals.singleton.sched.addEvent(this, msec);
    }
}
```

```
public class State {
    public Robot.Pose pose;
    public Board.Marks marks;

…

    public void drawAfter(int msec) {
        new Event(e -> {
            Globals.singleton.canvas.clear();
            Globals.singleton.robot.draw(pose);
            Globals.singleton.board.draw(marks);
            return 0;
        }).fireAfter(msec);
    }
}
```

```
//Generate a sequence of draw events
public class Command {
…
    static void moveToPose(Robot.Pose target) {
        State state = Globals.singleton.state;
        Robot.Pose pose   = new Robot.Pose(state.pose);
        Board.Marks marks = new Board.Marks(state.marks);

        int steps = 5;
        double delta = (target.base - pose.base) / steps;
        for(int i = 0; i < steps; i++, pose.base += delta)
            new State(pose, marks).drawAfter(50);

        delta = (target.arm1 - pose.arm1) / steps;
        for(int i = 0; i < steps; i++, pose.arm1 += delta)
            new State(pose, marks).drawAfter(50);

        delta = (target.arm2 - pose.arm2) / steps;
        for(int i = 0; i < steps; i++, pose.arm2 += delta)
            new State(pose, marks).drawAfter(50);
…
        Globals.singleton.state = state;
    }
```

```
public class EventScheduler implements Stoppable {
    Deque<Event> dq; //event queue
    long now;        //simulation time
…
    public EventScheduler() {
        dq = new ArrayDeque<>(); //event queue
        now = 0;
    }

    public void addEvent(Event e, int delayMs) {
        long t = dq.isEmpty() ? now : dq.peekLast().fireAt;
        e.fireAt = t + delayMs; //update event fire time
        dq.addLast(e);          //add to the event queue
    }

    //Runnable
    public void run() {
…
            long start = System.currentTimeMillis();
            while(!done) {
                Thread.sleep(1);
                if(done)
                    break;
```

```
…
                now = System.currentTimeMillis() - start;

                //run until queue is empty
                if(doneIfEmpty && dq.isEmpty())
                    break;

                if(!dq.isEmpty() &&
                    dq.peekFirst().fireAt <= now) { //if the time is mature
                    Event e = dq.removeFirst();
                    e.handler.apply(e);  //call the handler on this thread
                }
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("EventScheduler done");
    }
}
```

Game and Player

- Game
  - Creates two players
  - Synchronize the players' executions
  - Queue draw-events through commands
- Player
  - Blocked on a lock to synchronize
  - Pass its next mark position to callback
    - PlayerComputer computes the next position
    - PlayerHuman takes input from a keyboard

```
public interface Player extends Stoppable {
    public int getMark(); //O or X
    public void myTurn(Board.Marks marks,
                       Function<Integer,Integer> callback);
}


public class Game implements Stoppable {
    public Game() {
        o = new PlayerHuman(Board.mO);
        x = new PlayerComputer(Board.mX);
        lock = new Semaphore(1);
…
    }
…
    public void run() {
        try {
            Thread ot = new Thread(o); ot.start();
            Thread xt = new Thread(x); xt.start();
…
            Player curr = o;    //current player
            while(!done) {
                lock.acquire(); //released by callback
…
                final int mark = curr.getMark();
```

```
            curr.myTurn(new Board.Marks(Globals.singleton.state.marks),
                        i -> { //This callback works like a continuation
                    Command.mark(i, mark);

                    if(winner(Globals.singleton.state.marks) != Board.mN)
                        stop();
                    if(!hasEmpty(Globals.singleton.state.marks))
                        stop();

                    lock.release(); //unblock the Game thread
                    return 0;
                });
…
                //switch the current player
                curr = curr == o ? x : o;
            }

            Globals.singleton.stop(false/*immediately*/);
            o.stop(); x.stop();
            ot.join();  xt.join();
…
        }
```

```
public class PlayerComputer implements Player {
…
    //Player
    public void myTurn(Board.Marks marks,
                       Function<Integer,Integer> callback) {
        this.marks = marks;
        this.callback = callback;
        myturn.release();
    }
    //Runnable
    public void run() {
        done = false;
…
        while(!done) {
                myturn.acquire();
                if(done)
                    break;

                int i = nextMark(marks, myMark);
                callback.apply(i);
            }
        }
…
}
```

```
public class PlayerHuman implements Player {
…
    public void run() {
…
            while(!done) {
                myturn.acquire();
                if(done)
                    break;

                while(!done) {
                    char c = Globals.singleton.keyStrokes.remove();
                    if(c == 'q' || c == 'Q') {
                        Globals.singleton.stop(false/*immediately*/);
                        done = true;
                        break;
                    }
                    if('1' <= c && c <= '9') {
                        int i = c - '1';
                        if(marks.getMark(i) == Board.mN) {
                            callback.apply(i);
                            break;
                        }
                    }
                }
            }
…
}
```

Course Evaluation

Your feedback is important!

- Please submit your course evaluation at https://stonybrook.campuslabs.com/eval-home/

SUNY Korea