Topic 4.1: Concurrency Patterns: Monitors

Outline

Topic 4.1: Concurrency Patterns: Monitors
Monitors
FSP Models-to-Java Monitors
Producers/Consumers
Outline

Topic 4.1: Concurrency Patterns: Monitors

Monitors
FSP Models-to-Java Monitors
Producers/Consumers
About Monitors

- ‘passive’ shared objects: methods invoked by (active) threads
About Monitors

- ‘passive’ shared objects: methods invoked by (active) threads
- synchronized methods to exclusively access private variables.
About Monitors

- ‘passive’ shared objects: methods invoked by (active) threads
- synchronized methods to exclusively access private variables.
- *Condition Synchronization:* guarded execution:
About Monitors

• ‘passive’ shared objects: methods invoked by (active) threads
• synchronized methods to exclusively access private variables.
• *Condition Synchronization*: guarded execution:
  • looped guard-test containing `wait()`
About Monitors

- ‘passive’ shared objects: methods invoked by (active) threads
- synchronized methods to exclusively access private variables.
- *Condition Synchronization*: guarded execution:
  - looped guard-test containing `wait()`
  - `notify()`/`notifyAll()` after guard-state change
class Monitor
{
    private int x = 0;

    void addOne(int n, int delay) throws InterruptedException
    {
        System.out.println("Thread " + n + ": x is " + x);
        int t = x + 1;
        System.out.println("Thread " + n + ": t is " + t);
        Thread.sleep(delay);
        x = t;
        System.out.println("Thread " + n + ": x is " + x);
    }
}
Synchronisation Demo - I

class Monitor
{
    private int x = 0;

    synchronized
    void addOne(int n, int delay) throws InterruptedException
    {
        System.out.println("Thread " + n + ": x is " + x);
        int t = x + 1;
        System.out.println("Thread " + n + ": t is " + t);
        Thread.sleep(delay);
        x = t;
        System.out.println("Thread " + n + ": x is " + x);
    }
}
Synchronisation Demo - II

class T extends Thread
{
    Monitor obj;

    private int name, threadDelay;

    T(Monitor o, int n, int delay){
        obj = o;
        name = n;
        threadDelay = delay;
    }

    public void run(){
        try {obj.addOne(name, threadDelay); }
        catch (InterruptedException e) { }
    }
}
public class MonitorExample {

    public static void main(String[] args) throws InterruptedException {
        int delay = Integer.parseInt(args[0]);
        Monitor m = new Monitor();
        T t1 = new T(m, 1, delay);
        T t2 = new T(m, 2, delay/2);
        T t3 = new T(m, 3, delay/3);

        t1.start(); t2.start(); t3.start();

        Thread.sleep(10); m.addOne(0, 0);
    }
}
And what happens ... **without** synchronisation

```bash
$ java MonitorExample 20
Thread 1: x is 0
Thread 1: t is 1
Thread 2: x is 0
Thread 2: t is 1
Thread 3: x is 0
Thread 3: t is 1
Thread 3: x is 1
Thread 0: x is 1
Thread 0: t is 2
Thread 0: x is 2
Thread 2: x is 1
Thread 1: x is 1

$ java MonitorExample 30
Thread 1: x is 0
Thread 1: t is 1
Thread 2: x is 0
Thread 2: t is 1
Thread 3: x is 0
Thread 3: t is 1
Thread 0: x is 0
Thread 0: t is 1
Thread 0: x is 1
Thread 3: x is 1
Thread 2: x is 1
Thread 1: x is 1
```

And what happens ... without synchronisation

$ java MonitorExample 20
Thread 1: x is 0
Thread 1: t is 1
Thread 2: x is 0
Thread 2: t is 1
Thread 3: x is 0
Thread 3: t is 1
Thread 3: x is 1
Thread 0: x is 1
Thread 0: t is 2
Thread 0: x is 2
Thread 2: x is 1
Thread 1: x is 1

$ java MonitorExample 30
Thread 1: x is 0
Thread 1: t is 1
Thread 2: x is 0
Thread 2: t is 1
Thread 3: x is 0
Thread 3: t is 1
Thread 3: x is 0
Thread 0: x is 0
Thread 0: t is 1
Thread 0: x is 1
Thread 3: x is 1
Thread 2: x is 1
Thread 1: x is 1

$
And what happens ... with synchronisation

$ java MonitorExample 20
Thread 1: x is 0
Thread 1: t is 1
Thread 1: x is 1
Thread 0: x is 1
Thread 0: t is 2
Thread 0: x is 2
Thread 3: x is 2
Thread 3: t is 3
Thread 3: x is 3
Thread 2: x is 3
Thread 2: t is 4
Thread 2: x is 4

$
And what happens ... with synchronisation

```sh
$ java MonitorExample 20
Thread 1: x is 0
Thread 1: t is 1
Thread 1: x is 1
Thread 0: x is 1
Thread 0: t is 2
Thread 0: x is 2
Thread 3: x is 2
Thread 3: t is 3
Thread 3: x is 3
Thread 2: x is 3
Thread 2: t is 4
Thread 2: x is 4
$ Thread 1 runs to completion
Thread 0 runs to completion
Thread 3 runs to completion
Thread 2 runs to completion
no matter what delay
```
class Monitor
{
    private int x = 0;

    synchronized
    void addOne(int n, int delay) throws InterruptedException
    {
        while (!(x==n)) wait();
        System.out.println("Thread " + n + ": x is " + x);
        int t = x + 1;
        System.out.println("Thread " + n + ": t is " + t);
        Thread.sleep(delay);
        x = t;
        System.out.println("Thread " + n + ": x is " + x);
    }
}
Now with Conditional synchronisation ..

class Monitor
{
    private int x = 0;

    synchronized
    void addOne(int n, int delay) throws InterruptedException
    {
        while (!(x==n)) wait();
        System.out.println("Thread " + n + ": x is " + x);
        int t = x + 1;
        System.out.println("Thread " + n + ": t is " + t);
        Thread.sleep(delay);
        x = t;

        System.out.println("Thread " + n + ": x is " + x);
    }
}
class Monitor
{
    private int x = 0;

    synchronized
    void addOne(int n, int delay) throws InterruptedException
    {
        while (!(x==n)) wait();
        System.out.println("Thread " + n + ": x is " + x);
        int t = x + 1;
        System.out.println("Thread " + n + ": t is " + t);
        Thread.sleep(delay);
        x = t;
        notifyAll();
        System.out.println("Thread " + n + ": x is " + x);
    }
}
And this is what happens ...

```
$ java MonitorExample 20
Thread 0: x is 0
Thread 0: t is 1
Thread 0: x is 1
Thread 1: x is 1
Thread 1: t is 2
Thread 1: x is 2
Thread 2: x is 2
Thread 2: t is 3
Thread 2: x is 3
Thread 3: x is 3
Thread 3: t is 4
Thread 3: x is 4
$
```
And this is what happens ... 

```
$ java MonitorExample 20
Thread 0: x is 0
Thread 0: t is 1
Thread 0: x is 1
Thread 1: x is 1
Thread 1: t is 2
Thread 1: x is 2
Thread 2: x is 2
Thread 2: t is 3
Thread 2: x is 3
Thread 3: x is 3
Thread 3: t is 4
Thread 3: x is 4
$
```

We will always get:

- Thread 0 runs to completion
- Thread 1 runs to completion
- Thread 2 runs to completion
- Thread 3 runs to completion

no matter what delays
Outline

Topic 4.1: Concurrency Patterns: Monitors

Monitors
FSP Models-to-Java Monitors
Producers/Consumers
FSP Models to Java Monitors

FSP:

\[
\text{when (guard) actionA} \rightarrow P
\]

Java:

\[
synchronized \text{ type } \text{actionA} (...) \{
\text{while (!guard) \{ wait() \; \}}
\]

\[
// \text{Code for process P}
\]

When guard is changed, signal with \text{notify()}\.

Active objects initiate actions: implement as Threads

Passive (shared) objects respond to actions: implement as Monitors
Outline

Topic 4.1: Concurrency Patterns: Monitors
- Monitors
- FSP Models-to-Java Monitors
- Producers/Consumers
Producers and Consumers: the concept

- Producers: *produce* items which are sent to consumer(s)
Producers and Consumers: the concept

- Producers: *produce* items which are sent to consumer(s)
- Consumers: receive items and process them independently
Producers and Consumers: the concept

- Producers: *produce* items which are sent to consumer(s)
- Consumers: receive items and process them independently
- synchronous or buffered communication
Booth Street East Car Park — A Passive Object
CARPARKCONTROL(N=4) = SPACES[N],
SPACES[i:0..N] = (when(i>0) arrive->SPACES[i-1]
|when(i<N) depart->SPACES[i+1]
).

ARRivals  = (arrive->ARRIVALS).
DEPARTURES = (depart->DEPARTURES).

||CARPARK = (ARRIVALS||CARPARKCONTROL(4)||DEPARTURES).

FSP description
class CarParkControl {

    protected int spaces;
    protected int capacity;

    CarParkControl(int n) {
        capacity = spaces = n;
    }

    synchronized void arrive() throws InterruptedException {
        while (spaces==0) wait();
        --spaces;
        notifyAll();
    }

    synchronized void depart() throws InterruptedException{
        while (spaces==capacity) wait();
        ++spaces;
        notifyAll();
    }

}
CountDown Timer - An Active Object

COUNTDOWN (N=3) = ( start -> COUNTDOWN[N]),

COUNTDOWN[i:0..N]= ( when (i>0) tick -> COUNTDOWN[i-1]
| when (i==0) beep -> STOP
| stop -> STOP
).
Java for CountDown Timer - I

```java
public class CountDown extends Applet implements Runnable {
    Thread counter; int i;  // Applet methods
    final static int N = 3;
    AudioClip beepSound, tickSound;  // (don’t confuse with
    NumberCanvas display;  //    Thread start/stop )

    public void init() {...}  // required by Runnable;
    public void start() {...}  // called by Thread counter
    public void stop() {...}

    public void run() {...}  // local
    private void tick(){...}
    private void beep(){...}
```
public void start() { // method start for Applet
    counter = new Thread(this);
    i = N; counter.start(); // method start for Thread counter
}

public void stop() { // method stop for Applet
    counter = null; // (not using Thread stop method)
}

public void run() {
    while(true) {
        if (counter == null) return;
        if (i>0) { tick(); --i; }
        if (i==0) { beep(); return;}
    }
}