Lab Management
An Example

Laboratorio di Reti
2014/2015
Prof. Laura Ricci

Speaker: Alessandro Lulli - lulli@di.unipi.it

Esercizio: Gestione Laboratorio

ESERCIZIO: GESTIONE LABORATORIO

Il laboratorio di Informatica del Polo Marzotto è utilizzato da tre tipi di utenti, studenti, tesisti e professori ed ogni utente deve fare una richiesta al tutor per accedere al laboratorio. I computers del laboratorio sono numerati da 1 a 20. Le richieste di accesso sono diverse a seconda del tipo dell'utente:

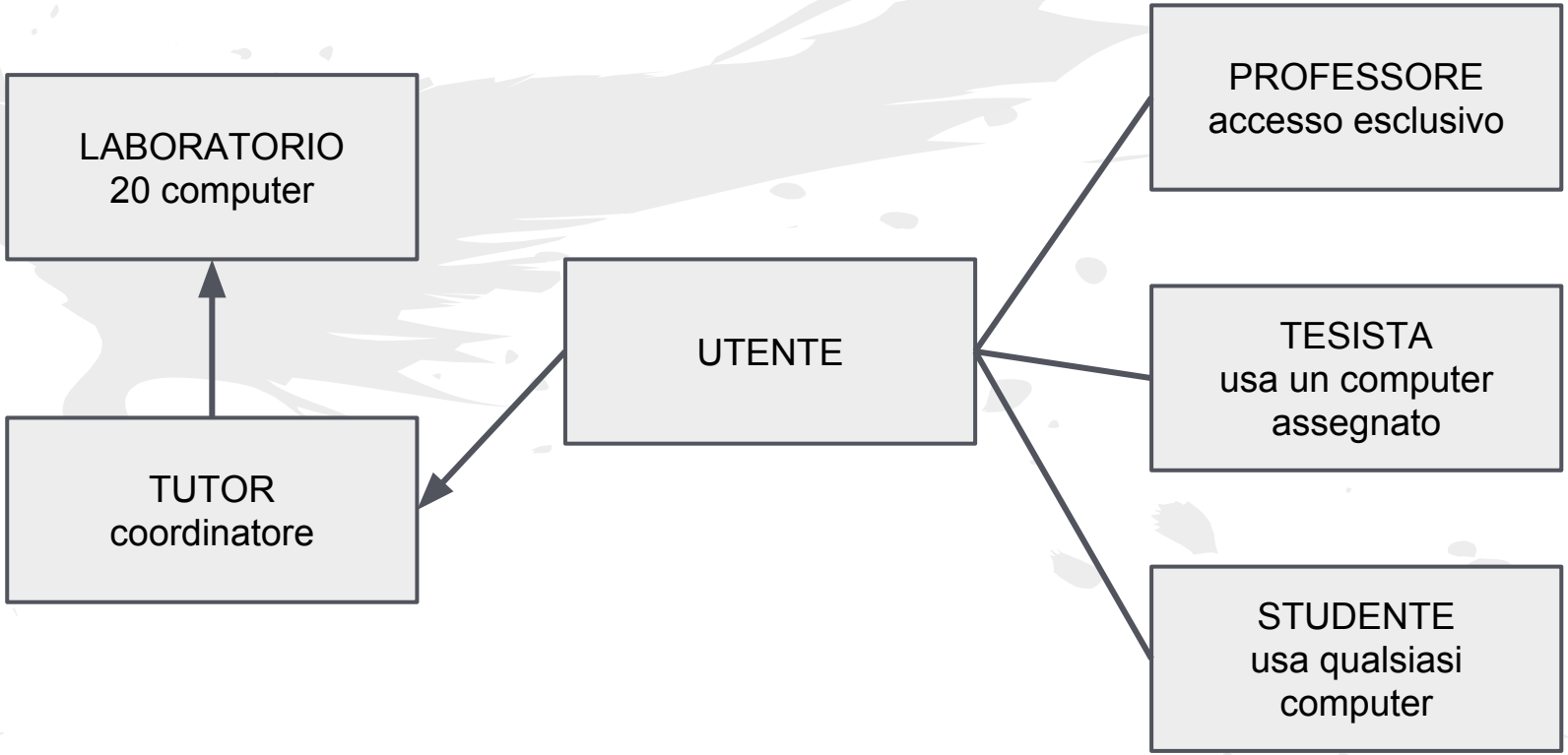
a) i professori accedono in modo esclusivo a tutto il laboratorio, poichè hanno necessità di utilizzare tutti i computers per effettuare prove in rete.

b) i tesisti richiedono l'uso esclusivo di un solo computer, identificato dall'indice i, poichè su quel computer è istallato un particolare software necessario per lo sviluppo della tesi.

c) gli studenti richiedono l'uso esclusivo di un qualsiasi computer.

I professori hanno priorità su tutti nell'accesso al laboratorio, i tesisti hanno priorità sugli studenti. (prosegue nella pagina successiva)

Soggetti del programma



Il laboratorio

- gestisce lo stato dei computer: in uso o liberi
- caso speciale se un professore è all'interno del laboratorio

```
public class Lab
{
    final int _pcNumber;
    boolean _prefessorIsIn;
    Person[] _pc;

    public Lab(final int pcNumber_)
    {
        _pcNumber = pcNumber_;
        _prefessorIsIn = false;
        _pc = new Person[_pcNumber];
    }

    public int getAvailablePC()
    {
        if (_prefessorIsIn)
        {
            return -1;
        }
        for (int i = 0; i < _pc.length; i++)
        {
            if (_pc[i] == null)
            {
                return i;
            }
        }
        return -1;
    }
```

Il tutor

- gestisce il laboratorio lo **stato condiviso** tra gli utenti
- un lock per accedere in maniera esclusiva allo stato dei computer
- tre condizioni:
  - attesa per studenti
  - attesa per professori
  - attesa per usare un computer (per tesisti)

```
public class Tutor
{
    final Lab _lab;
    final ReentrantLock _lock;
    final Condition _studentCondition;
    final Condition _profCondition;
    final Condition[] _conditions;

    public Tutor(final Lab l)
    {
        _lab = l;
        _lock = new ReentrantLock();
        _studentCondition = _lock.newCondition();
        _profCondition = _lock.newCondition();
        _conditions = new Condition[_lab._pcNumber];
        for (int i = 0; i < _conditions.length; i++)
        {
            _conditions[i] = _lock.newCondition();
        }
    }
```

Utenti del laboratorio

- il laboratorio di Informatica del Polo Marzotto è utilizzato da tre tipi di utenti:
  - studenti
  - tesisti
  - professori
- ogni utente accede *k* volte al laboratorio, con *k* generato casualmente

```
public abstract class Person implements Runnable {

    int k; // Number of access to lab
    final Tutor tutor;

    public Person(final Tutor t) {
        k = (int)(Math.random()*10);
        tutor = t;
    }

    public void work(){
        sleep();
        k--;
    }

    public void sleep(){
        final int execTime = (int)(Math.random()*1000);
        try{
            Thread.sleep(execTime);
        }catch(final InterruptedException e){}
    }

}
```

Professore

- i professori accedono in modo esclusivo a tutto il laboratorio, poichè hanno necessità di utilizzare tutti i computers per effettuare prove in rete
- i professori hanno priorità su tutti nell'accesso al laboratorio

```
public class Professor extends Person {

    public Professor(final Tutor t) {
        super(t);
    }

    @Override
    public void run() {
        for(int i=0;i<k;i++){
            tutor.askLab(this);
            work();
            tutor.releaseLab(this);
            sleep();
        }
    }

}
```

Professore: accesso laboratorio

- quando il laboratorio è tutto libero (*isEmpty()*) accedono a tutto il laboratorio (*set(prof)*)

```
public void askLab(final Professor prof)
{
    _lock.lock();
    try
    {
        try
        {
            while (!_lab.isEmpty())
            {
                _profCondition.await();
            }
            _lab.set(prof);
        } catch (final InterruptedException e)
        {
            e.printStackTrace();
        }
    } finally
    {
        _lock.unlock();
    }
}
```

```
public void set(final Professor p)
{
    _prefessorIsIn = true;
    printStatus();
}
```

Professore: uscita laboratorio

- quando un professore esce:
  - fa entrare un altro professore se in attesa *lock.hasWaiters*
  - altrimenti sveglia studenti

```
public void releaseLab(final Professor p)
{
    _lock.lock();
    try
    {
        _lab.unSet(p);
        if (_lock.hasWaiters(_profCondition))
        {
            _profCondition.signal();
        } else
        {
            for (int i = 0; i < _conditions.length; i++)
            {
                _conditions[i].signal();
            }
            _studentCondition.signalAll();
        }
    } finally
    {
        _lock.unlock();
    }
}
```

Tesista

- i tesisti richiedono l'uso esclusivo di un solo computer, identificato dall'indice *pc*, poichè su quel computer è istallato un particolare software necessario per lo sviluppo della tesi

```
public class Tesista extends Person {
    int pc;

    public Tesista(final Tutor t) {
        super(t);
        pc = (int)(Math.random()*20);
    }

    @Override
    public void run() {
        for(int i=0;i<k;i++){
            tutor.askLab(this);
            work();
            tutor.releaseLab(this);
            sleep();
        }
    }

    public int GetPC(){
        return pc;
    }
}
```

Tesista: accesso al laboratorio

- i tesisti hanno priorità sugli studenti, ma non sui professori
- accedono se:
  - il pc richiesto è disponibile
  - non ci sono professori in attesa

```
public void askLab(final Tesista tes)
{
    final int pc = tes.GetPC();
    _lock.lock();
    try
    {
        try
        {
            while (!_lab.isAvailable(pc) || _lock.hasWaiters(_profCondition))
            {
                _conditions[pc].await();
            }
            _lab.set(tes);
        } catch (final InterruptedException e)
        {
            e.printStackTrace();
        }
```

Tesista: uscita laboratorio

- libera la postazione con *unSet*
- se professori in attesa prova a svegliarne uno

```
public void releaseLab(final Tesista t)
{
    _lock.lock();
    try
    {
        final int pc = t.GetPC();
        _lab.unSet(t);
        if (_lock.hasWaiters(_profCondition))
        {
            _profCondition.signal();
        } else if (_lock.hasWaiters(_conditions[pc]))
        {
            _conditions[pc].signal();
        } else
        {
            _studentCondition.signal();
        }
    } finally
    {
        _lock.unlock();
    }
}
```

```
public void unSet(final Tesista t)
{
    _pc[t.GetPC()] = null;
    printStatus();
}
```

Studente

- gli studenti richiedono l'uso esclusivo di un qualsiasi computer

```
public class Student extends Person {

    int pc = -1;

    public Student(final Tutor t) {
        super(t);
    }

    @Override
    public void run() {
        for(int i=0;i<k;i++){
            pc = tutor.askLab(this);
            work();
            tutor.releaseLab(this);
            pc = -1;
            sleep();
        }
    }

    public int GetPC(){
        return pc;
    }

}
```

Studente: accesso al laboratorio

- in attesa:
  - se non ci sono computer liberi *isFull*
  - c'è un professore in attesa

```
public int askLab(final Student student)
{
    int pc = -1;
    _lock.lock();
    try
    {
        try
        {
            while (_lab.isFull() || _lock.hasWaiters(_profCondition))
            {
                _studentCondition.await();
            }
            pc = _lab.getAvailablePC();
            _lab.set(student, pc);
        } catch (final InterruptedException e)
        {
            e.printStackTrace();
        }
    } finally
    {
        _lock.unlock();
    }
    return pc;
}
```

Studente: uscita laboratorio

- quando esce:
  - libera il pc che ha utilizzato
  - sveglia un professore se in attesa
  - sveglia un tesista
  - sveglia un altro studente

```
public void releaseLab(final Student p)
{
    _lock.lock();
    try
    {
        final int pc = p.GetPC();
        _lab.unSet(p);
        if (_lock.hasWaiters(_profCondition))
        {
            _profCondition.signal();
        } else if (_lock.hasWaiters(_conditions[pc]))
        {
            _conditions[pc].signal();
        } else
        {
            _studentCondition.signal();
        }
    } finally
    {
        _lock.unlock();
    }
}
```

Main class

```
public static void main(final String[] args) {
    final int n = 23;
    final int m = 7;
    final int k = 2;

    final ThreadPoolExecutor executor = (ThreadPoolExecutor) Executors.newCachedThreadPool();
    final Lab lab = new Lab();
    final Tutor tutor = new Tutor(lab);

    for(int i=0;i<m;i++){
        final Tesista t = new Tesista(tutor);
        executor.execute(t);
    }

    for(int i=0;i<n;i++){
        final Student s = new Student(tutor);
        executor.execute(s);
    }

    for(int i=0;i<k;i++){
        final Professor p = new Professor(tutor);
        executor.execute(p);
    }
```

Output example

- quando un professore è nel laboratorio non ci sono altri utenti
- studenti e tesisti condividono il laboratorio

```
00000100100101100000| St: 5   | Te: 0   | Prof:0   |
00000000100101100000| St: 4   | Te: 0   | Prof:0   |
00000000100100100000| St: 3   | Te: 0   | Prof:0   |
00000000100000100000| St: 2   | Te: 0   | Prof:0   |
00000000000000100000| St: 1   | Te: 0   | Prof:0   |
00000000000000000000| St: 0   | Te: 0   | Prof:0   |
11111111111111111111| St: 0   | Te: 0   | Prof:1   |
00000000000000000000| St: 0   | Te: 0   | Prof:0   |
00010000000000000000| St: 0   | Te: 1   | Prof:0   |
10010000000000000000| St: 1   | Te: 1   | Prof:0   |
11010000000000000000| St: 2   | Te: 1   | Prof:0   |
```