

Sistemi Operativi

(modulo di Informatica II)

Laboratorio

Sincronizzazione in Java (**Lock e variabili condizione in Java**)

Patrizia Scandurra

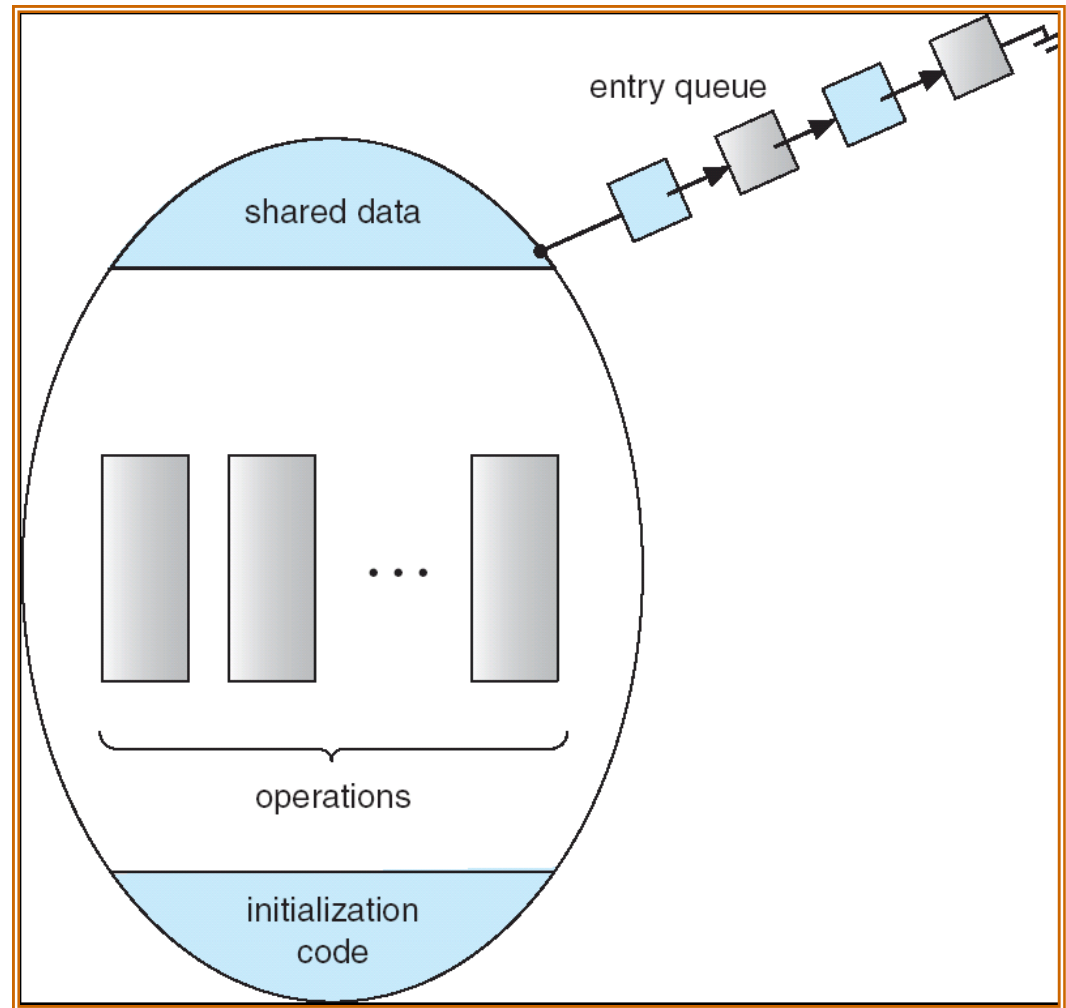
Università degli Studi di Bergamo

Visione schematica dei monitor

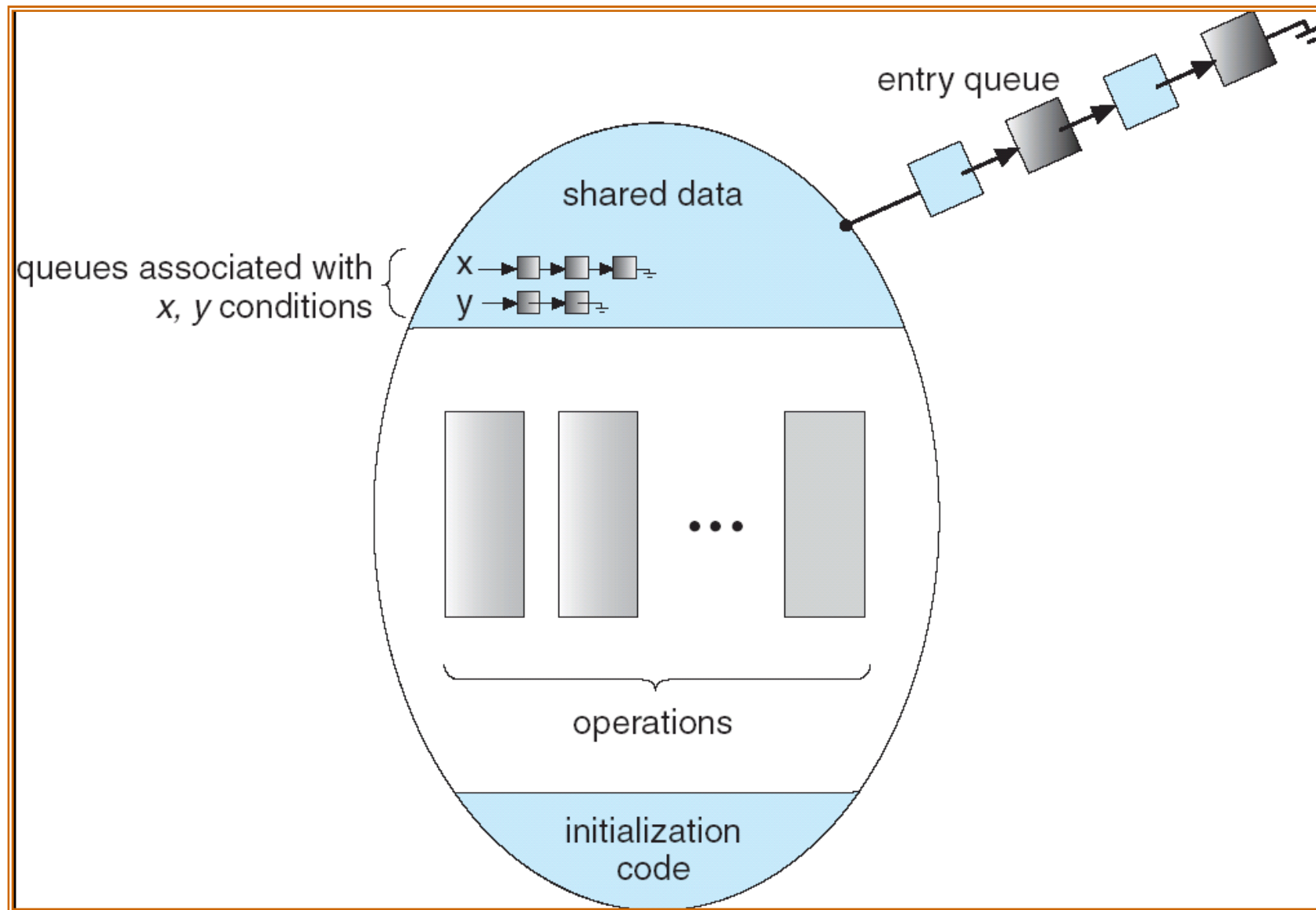
MONITOR è un costrutto linguistico

- **in Java:** classi con metodi sincronizzati

```
class monitor-name {  
    // variable declarations  
    synchronized public  
    entry p1(...) {  
        ...  
    }  
    synchronized public  
    entry p2(...) {  
        ...  
    }  
}
```



Monitor con le variabili “condizioni”



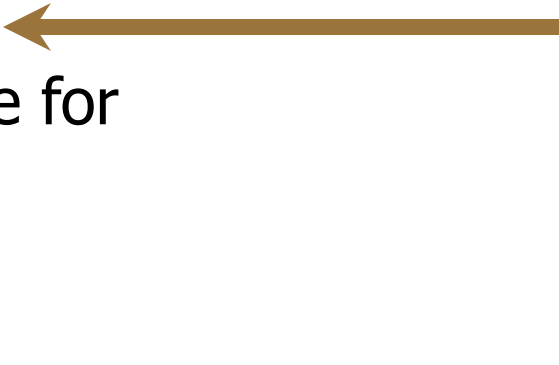
Mutua esclusione mediante lock e variabili condizione

Per realizzare i monitor con variabili condizione, da Java 5.0 si può utilizzare esplicitamente il concetto di oggetti **lock** e **variabili condizione**,

- mediante le interfacce **Lock** e **Condition** e
- la classe **ReentrantLock** in **java.util.concurrent.locks**

L'interfaccia Lock

```
package java.util.concurrent.locks;  
public interface Lock {  
    public void lock(); // Wait for the lock to be acquired  
    public void unlock();  
    public Condition newCondition();  
    // Create a new condition variable for  
    // use with the Lock.  
    ...  
}
```



- Le variabili condizione sono associate ad un lock!
- Non è possibile mettersi in attesa su una condition variable se non ho prima acquisito il Lock Object a cui è associata

L'interfaccia Condition

```
package java.util.concurrent.locks;  
public interface Condition {  
    public void await()  
        throws InterruptedException;  
    // Atomically releases the associated lock  
    // and causes the current thread to wait.  
    public void signal();  
    // Wake up one waiting thread.  
    public void signalAll();  
    // Wake up all waiting threads.  
    ...  
}
```

La classe ReentrantLock

```
package java.util.concurrent.locks;  
public class ReentrantLock implements Lock {  
    public ReentrantLock();  
    ...  
    public void lock();  
    public void unlock();  
    public Condition newCondition();  
    // Create a new condition variable and  
    // associated it with this lock object.  
}
```

Uso dei lock di Java.util.concurrent

E' a carico del programmatore, l'utilizzo dei metodi **lock()** e **unlock()** su un oggetto lock **per garantire la mutua esclusione**

In Java:

//creazione del lock

```
Lock key = new ReentrantLock();
```

```
key.lock();
```

//Mutua esclusione

```
try {
```

```
    ... // sezione critica: accedi alle risorse protette dal lock  
}
```

```
finally { key.unlock(); }
```


Le variabili “condizioni” di Java.util.concurrent -- creazione

In Java:

- Per **creare una variabile condizione** occorre prima creare un oggetto lock **ReentrantLock** e poi invocare su di esso il metodo **newCondition()**:

```
Lock key = new ReentrantLock();  
Condition condVar = key.newCondition();
```

- Dopo, sarà possibile invocare su “condVar” i metodi **await()**, **signal()** e **signalAll()**

Le variabili “condizioni” di Java.util.concurrent -- uso

Due operazioni possibili su una variabile “condition” *x*:

- *x.await()* – il thread che la invoca **viene sospeso nella coda di attesa di x**, con rilascio automatico del lock
- *x.signal()* – risveglia **arbitrariamente esattamente un thread della coda di x** se ce ne sono, altrimenti nulla; automaticamente, il thread svegliato compete con eventuali altri thread per riacquisire il lock
- *x.signalAll()* – **risveglia tutti thread della coda di x** se ce ne sono, altrimenti nulla; automaticamente, il thread svegliato compete con eventuali altri thread per riacquisire il lock

N.B.: I metodi classici *wait()* e *notify()/notifyall()* sono simili a *await()* e *signal()/signalAll()*, ma senza variabili “condizioni” con nome x,y, ecc..

Lock e variabili “condizioni” di Java.util.concurrent – uso combinato

In Java:

//creazione del lock e della condition

Lock key = new ReentrantLock();

Condition x = key.newCondition();

Condition y = key.newCondition(); ...

key.lock();

//Mutua esclusione

try {

//sezione critica: accedi alle risorse protette dal lock key e

// a x,y, ecc. con wait e/o signal()/signalAll

 while (condX) x.await();

 ...

 y.signal();

 }

finally { key.unlock(); }

Esempio del Producer-Consumer

- Vedi **<boundedbuffer>** con lock e variabili condizione

Esercizio

● Coke machine

- Usando i Lock e le Condition variable del package `java.util.concurrent.locks` come meccanismo di sincronizzazione, si realizzi in Java una soluzione al problema di prelevare lattine di coca-cola da una macchinetta e di rifornirla nel caso in cui rimanga vuota.
- Definire le classi per i thread Utente e Fornitore. Completare, inoltre, la classe `CokeMachine` (riportata sotto) contenente le lattine di coca-cola (oggetti condivisi) ed i metodi:
 - `remove(...)`, eseguito dal generico utente per prelevare una lattina dalla macchinetta;
 - `deposit(...)`, eseguito dal fornitore del servizio per caricare la macchinetta nel caso in cui rimane vuota.
- Si assumi che inizialmente la macchinetta è piena, e che un utente (a scelta: il primo a trovare la macchinetta vuota o l'utente che preleva l'ultima lattina) segnali al fornitore che la macchinetta è vuota

```
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

public class CokeMachine {

    private static final int N = 50; //Capacità della macchinetta

    private int count; //Numero effettivo di lattine presenti nella macchinetta
    private final Lock lock = new ReentrantLock(); //Lock per la mutua esclusione

    //Condition variables

    // .... <COMPLETARE>....

}
```