SISTEMI OPERATIVI

(MODULO DI INFORMATICA II)

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

Prof. Luca Gherardi

Prof.ssa Patrizia Scandurra (anni precedenti)

Università degli Studi di Bergamo

a.a. 2012-13

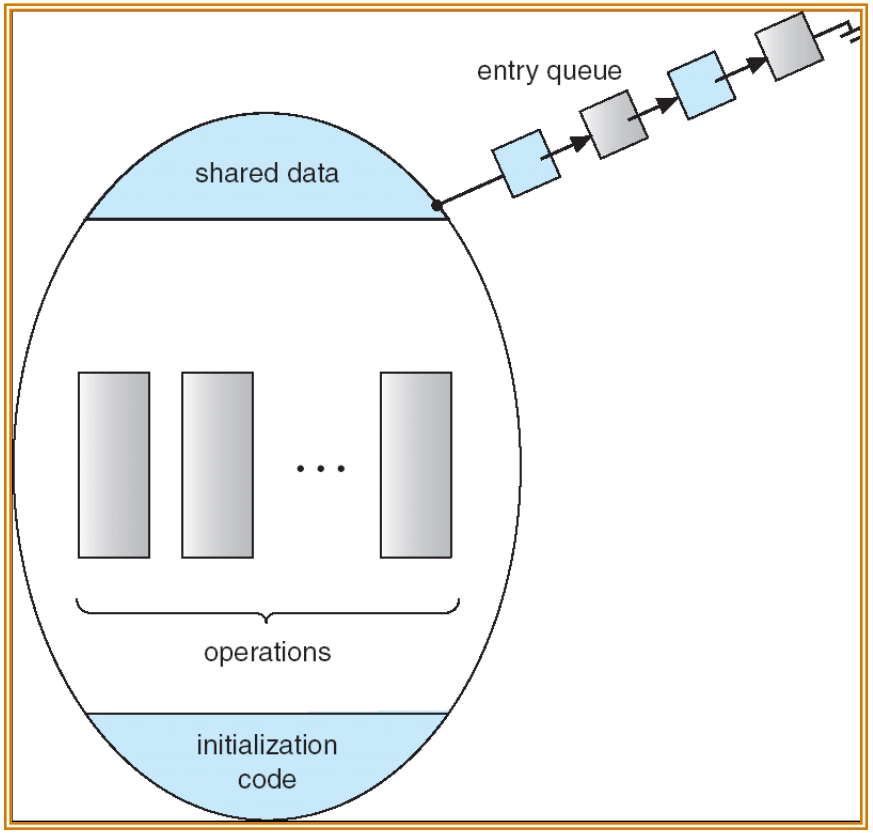
La gestione dei thread in Java

- Creazione e avvio
- Terminazione e cancellazione dei thread
- La gestione dei segnali
- **Sincronizzazione**
  - Monitor e **variabili condivise**
- Dati specifici dei thread
- Schedulazione
- Gruppi di thread

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(…) {
       …
     }
}
```



Mutua esclusione: lock

Per la mutua esclusione

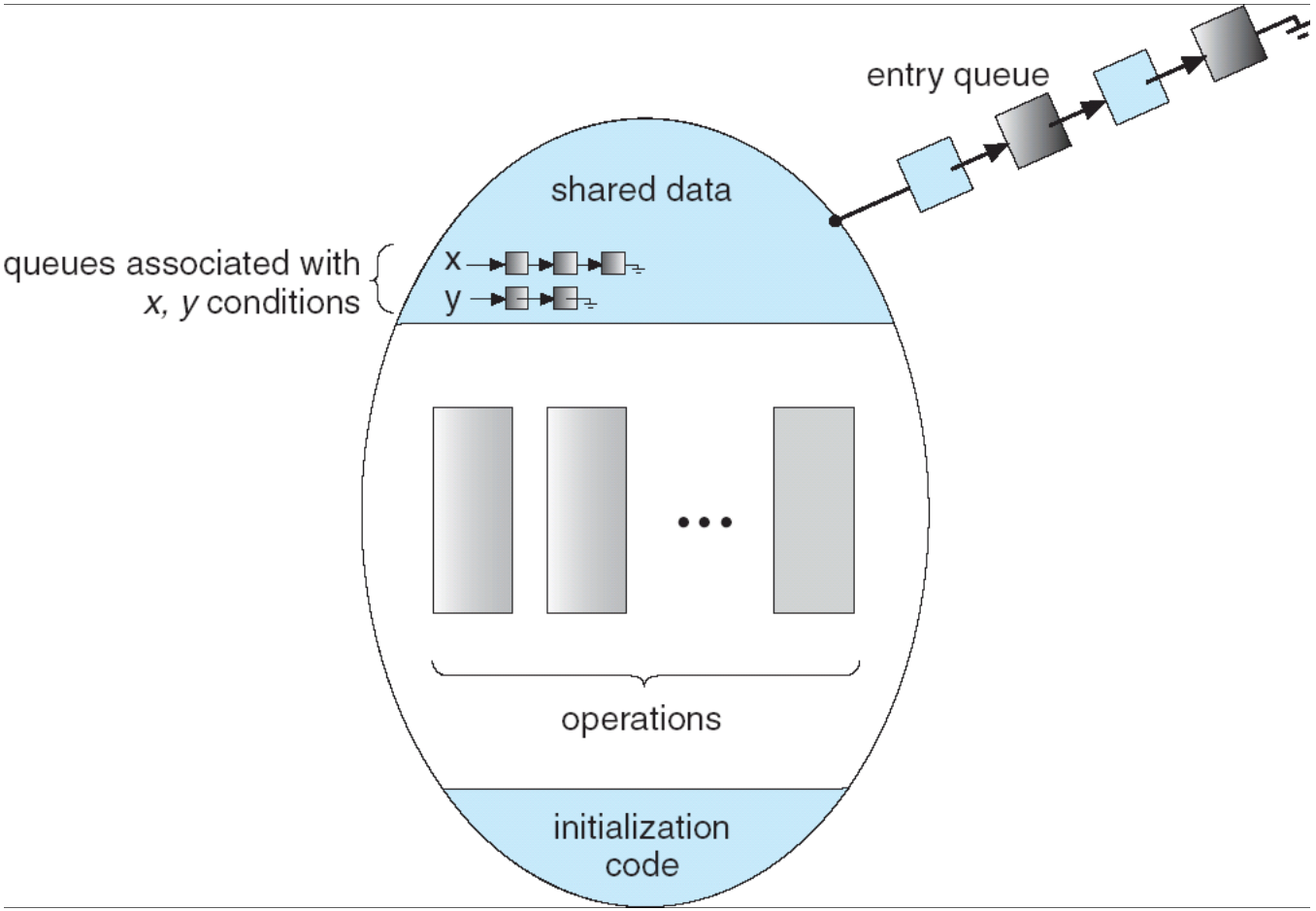
- oltre ai metodi/blocchi synchronized
- In Java si può utilizzare esplicitamente il concetto di **lock e condizioni**
  - Verranno introdotte nell'esercitazione di oggi

Le variabili “condizioni” (*condition*)

- Nozioni teoriche

- `Condition x,y;`
- Due operazioni possibili su una variabile “condition”:
  - `x.wait()`: il thread che la invoca viene sospeso nella coda di x
  - `x.signal()`: risveglia *esattamente* un thread in attesa nella coda di x se ce ne sono, altrimenti non fa nulla

Monitor con le variabili “condizioni”



Le variabili “condizioni” (*condition*)

- In Java

- Introdotte in Java5 nel package `java.util.concurrent`
  - mediante le interfacce `Lock` e `Condition`
  - e la classe `ReentrantLock`
  - definite 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 Condition newCondition();
    // Create a new condition variable for
    // use with the Lock.
  public void unlock();
  …
}
```

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 Condition newCondition();
    // Create a new condition variable and
    // associated it with this lock object.
  public void unlock();
}
```

Java.util.concurrent e i monitor

- Non si usa la keyword **synchronized**
- Resta a carico del programmatore, l'utilizzo dei metodi `lock()` e `unlock()` dell'oggetto "key" per garantire la *mutua esclusione*:

```
//creazione del lock
Lock key = new ReentrantLock();
key.lock();
//Mutua esclusione
try {
   … // sezione critica: accedi alle risorse protette dal
lock
      }
finally { key.unlock(); }
```

java.util.concurrent e le variabili “condizioni” Creazione

- Ad ogni variabile **condizione** deve essere associato un **lock**
- Quindi, per creare una variabile Condition occorre prima creare un “oggetto lock” `ReentrantLock` e poi invocare su di esso il metodo `newCondition()`:

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

- Successivamente, sarà possibile invocare su `condVar` i metodi `await()` (con rilascio automatico del lock), `signal()` e `signalAll()`

java.util.concurrent e le variabili “condizioni” Uso

- `Condition x, y;`
- Tre operazioni possibili su una variabile “condition”:
  - `x.await()`: il thread che la invoca viene sospeso
  - `x.signal()`: risveglia *esattamente* un thread in attesa nella coda di x se ce ne sono, altrimenti nulla
  - `x.signalAll()`: risveglia tutti thread in attesa nella coda di x se ce ne sono, altrimenti nulla
- **N.B.:**
  - I metodi classici `wait()` e `notify()/notifyall()` corrispondono alle primitive `await()` e `signal()/signalAll()` sopra citati, ma senza l’uso delle variabili “condizioni”
  - Ad es. quando un thread riceve la callback di una `notify`, non riceve informazioni sul motivo (x, y, …) per cui è stato svegliato

java.util.concurrent e le variabili “condizioni” Uso nei monitor

```
//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: si accede alle risorse protette
    // dal lock key e a x,y, ecc. con wait e/o
    // signal()/signalAll
    while (condX) x.await(); // mi metto in attesa del
    …                        // verificarsi di condX
    y.signal();
    }
finally { key.unlock(); }
```

Programma della lezione

- Esempio del **Producer consumer con bounded buffer** implementato con monitor e variabili condizione

- Finire gli esercizi delle lezioni precedenti
- E completare gli esercizi di questa esercitazione