Quarta esercitazione

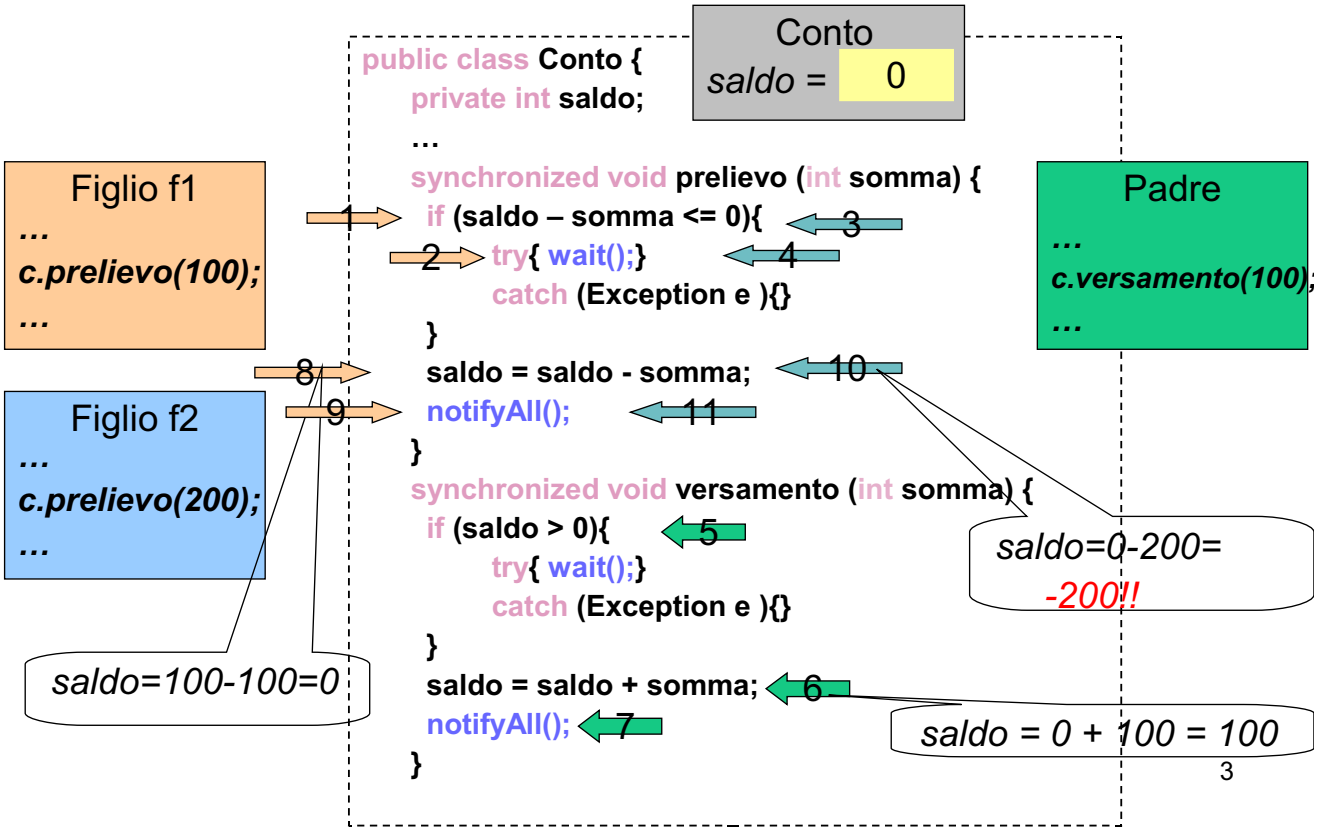
Java Thread

Stefano Monti
smonti@deis.unibo.it

Utilizzo di wait/notifyAll

- wait() e notifyAll() vanno
  - ***invocati nel codice dell'oggetto condiviso***
  - all'interno di metodi **synchronized** (dal momento che operano sul lock dell'oggetto stesso)
- metodo wait()
  - viene tipicamente utilizzato quando l'esecuzione di una sequenza di operazioni è possibile solo se l'oggetto si trova in un ben determinato stato (insieme dei valori delle variabili istanza)
  - e' necessario progettare attentamente la condizione di esecuzione del metodo wait(), in particolare:
    - che condizione devo testare?
    - all'uscita da una wait (a seguito di notifyAll()) devo verificare nuovamente la condizione di esecuzione della wait()?

Utilizzo di wait/notifyAll

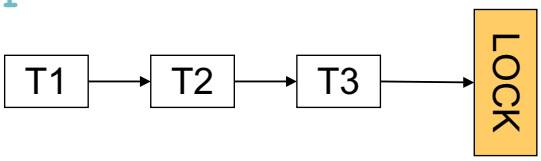


Utilizzo di wait/notifyAll

- In Java esiste (un solo lock, e quindi) una sola coda associata a ciascun oggetto.
- Tutti i Thread in attesa sul lock vengono inseriti nella stessa coda (l'ordine di accodamento non segue necessariamente quello di inserimento).
- I Thread in coda potrebbero essere bloccati **in metodi differenti** ed **attendere che si verifichino condizioni differenti**

Condizioni di sincronizzazione (1/2)

```
public synchronized void m1 () {
  if (condA) wait();
  //Operazioni
}
public synchronized void m2 () {
  if (condB) wait();
  //Operazioni
}
```



- La notifyAll() sblocca tutti i Thread, ma…

…verosimilmente **la condizione di proseguimento sarà verificata solo per alcuni di essi!!**

Condizioni di sincronizzazione (2/2)

- Tipicamente il metodo wait()
  - è preceduto dal controllo della condizione abilitante
  - è seguito da un nuovo controllo della condizione abilitante

```
while (cond){
wait();
}
```

  - sospende l'esecuzione dei Thread (il Thread che lo invoca **rilascia il lock!!!**)

Deadlock

- Supponiamo di avere due metodi:

```
public synchronized void m1 () {
  while (condA) wait();
  //Modifica dello stato
   notifyAll();
}
public synchronized void m2 () {
 while (condB) wait();
 //Modifica dello stato
   notifyAll();
}
```

- se **condA == true && condB == true** --> tutti i thread eseguono wait(), quindi si bloccano in attesa di una modifica dello stato **che non avverrà mai**, perché tutti i thread sono bloccati --> **deadlock!**

Deadlock - esempio

Con riferimento alla esercitazione precedente:

```
public synchronized void prelievo (int somma){
 while (saldo-somma<0)
   {try{wait();}catch(InterruptedException e){}}
  saldo-=somma;
  notifyAll();
}

public synchronized void versamento(int somma){
 while (saldo>0)
    try{wait();}catch(InterruptedException e){}
 saldo+=somma;
 notifyAll();
}
```

Figlio f1
```
…
c.prelievo(100);
…
```

Figlio f2
```
…
c.prelievo(60);
…
```

Padre
```
…
c.versamento(100);
…
```

Per una data sequenza di attivazione dei thread (es. f1,p,f2) potremmo avere saldo = 40 --> le condizioni per bloccare prelievi e versamenti sono entrambe verificate --> deadlock!

Ordinamento di Thread (1/2)

A default, nessun ordine sull'acquisizione del lock su un oggetto condiviso tra thread.

Supponiamo di necessitare di un **ordinamento FIFO (First In First Out)** tra Thread, cioè:

- N thread competono per l'accesso ad una risorsa condivisa
- i thread vengono messi in esecuzione in ordine di arrivo

Una soluzione possibile:

- oggetto CodaFIFO condiviso fra i thread che vogliamo ordinare
- espone due metodi
  - ***acquire()*** : il thread chiamante chiede di entrare in coda
  - ***release()*** : il thread chiamante chiede di uscire dalla coda

Ordinamento di Thread (2/2)

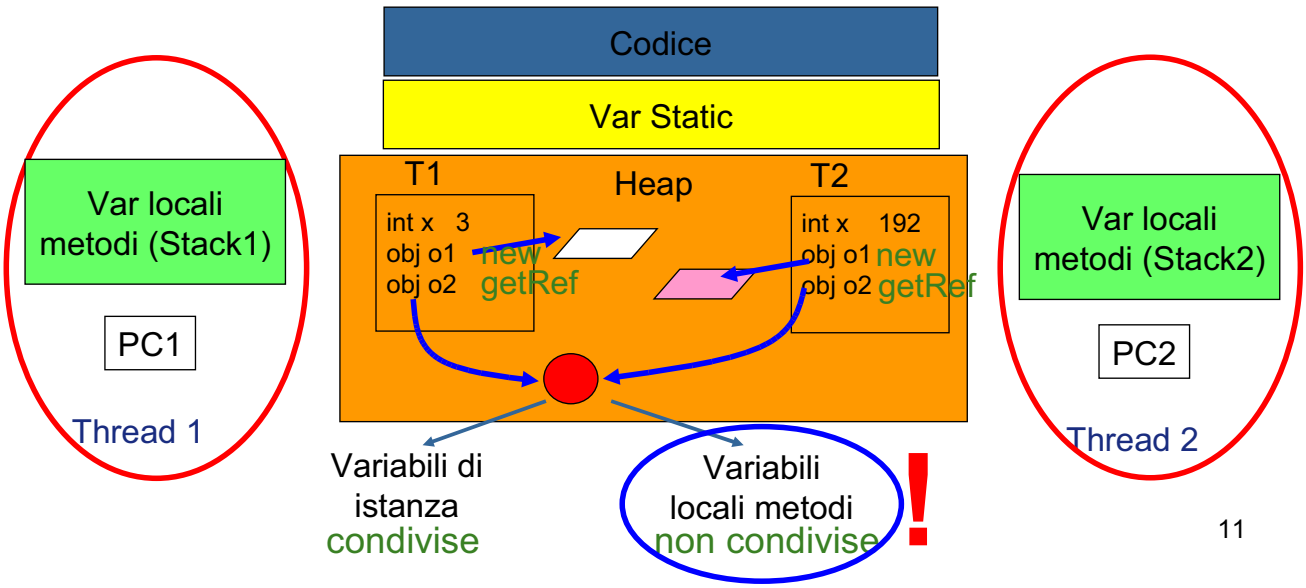
- Quindi, lato Thread:

```
public class MyThread extends Thread{
   CodaFIFO sharedQueue;
   ...
   public void run(){
   sharedQueue.acquire();
   //logica applicativa del thread
   sharedQueue.release();
   }
   }
```

Da ricordare…

*Le **variabili locali ai metodi dell'oggetto condiviso** vengono allocate sullo Stack e pertanto*

***non sono condivise tra i Thread** che accedono!!*



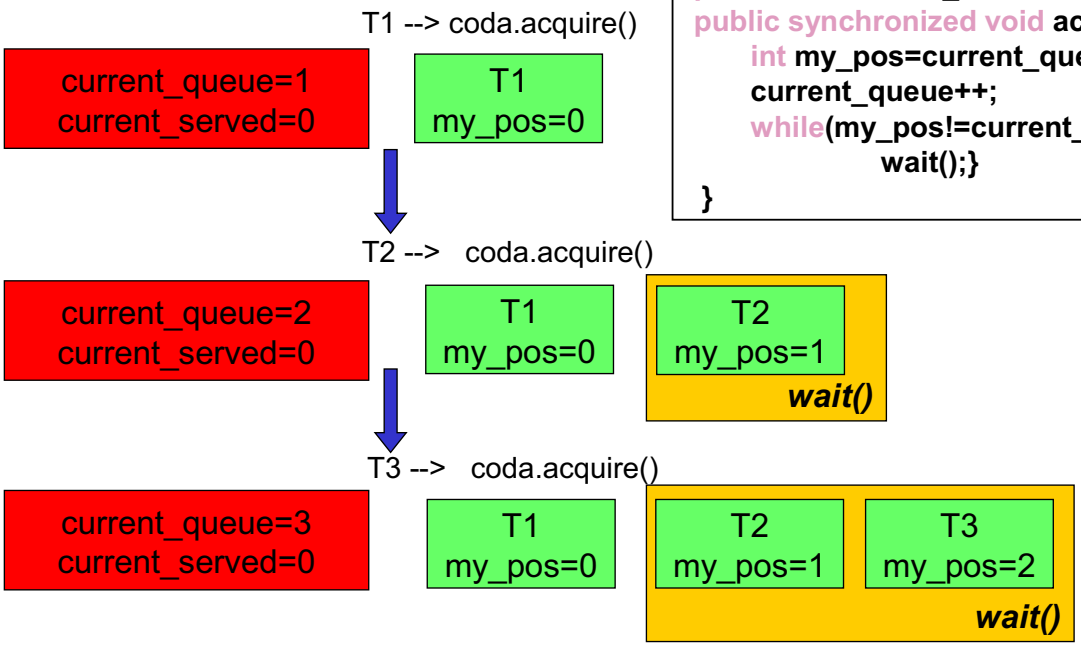
Coda FIFO

```
public class CodaFIFO{
  private int current_queue=0;//Rich servite in coda
  private int current_served=0;//Rich in servizio

  public synchronized void acquire(){
   int my_pos=current_queue;//Posizione del thread
  corrente nella coda
      current_queue++;
      try{ while(my_pos!=current_served){ //Condizione
      wait();} }
   catch(InterruptedException e){}
  }
   public synchronized void release(){
      current_served++;
      notifyAll();}
}
```

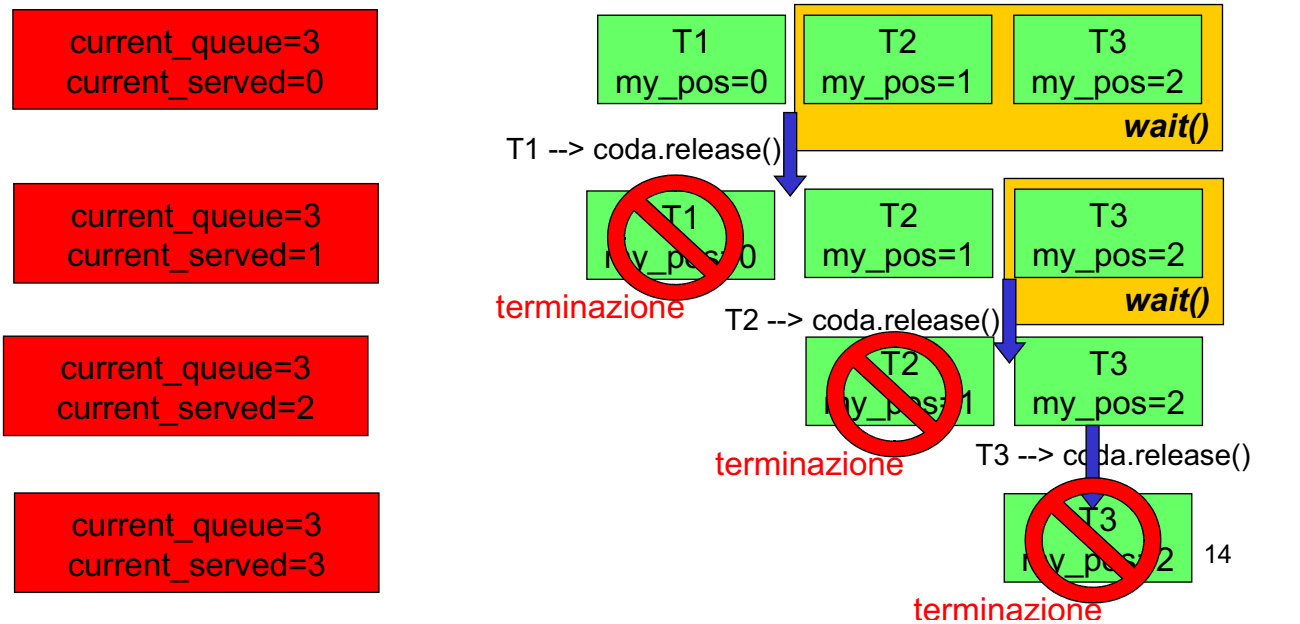
Coda FIFO

```
public class CodaFIFO{
 private int current_queue=0;
 private int current_served=0
 public synchronized void acquire(){
       int my_pos=current_queue;
       current_queue++;
       while(my_pos!=current_served){
                wait();}
 }
```



Coda FIFO

```
public class CodaFIFO{
  private int current_queue=0;
  private int current_served=0;
…
public synchronized void release(){
      current_served++;
      notifyAll();}
```



Gestione Code Distinte

```
public class Code{
  private int current_queue1=0;  //Rich servite in coda 1
  private int current_served1=0; //Rich in servizio in coda 1
  private int current_queue2=0;  //Rich servite in coda 2
  private int current_served2=0; //Rich in servizio in coda 2

  public synchronized void acquire_coda1(){
   int my_pos=current_queue1;//Posizione del thread corrente
  nella coda
      current_queue1++;
      try{ while(/*Condizione*/){ wait()
       . . .
 public synchronized void acquire_coda2(){
   int my_pos=current_queue2;//Posizione del thread corrente
  nella coda
      current_queue2++;
      try{ while(/*Condizione*/){ wait()
```


Esercizio

Realizzare un sistema di stampa di documenti

- due tipologie di documenti: **testi** e **immagini**
- i testi hanno **priorità maggiore** rispetto alle immagini: una **immagine viene accettata** per la stampa **solamente nel caso in cui non ci siano testi** in coda
- ciascuna coda deve essere gestita secondo una politica **FIFO**
- Il buffer della stampante ha capienza massima limitata, pari a **N_MAX** documenti. Solo dopo che N_MAX documenti sono stati accettati per la stampa, la stampante procede a stamparli

Esercizio - estensione

- il sistema attribuisce a ogni documento un **istante di stampa previsto**, calcolato sommando i tempi di stampa dei documenti **precedentemente accettati**
  - tempo di stampa testo: 7 secondi,
  - tempo di stampa immagine: 15 secondi

- **Al termine** della stampa di ogni documento, il sistema di stampa deve stampare a video l'istante di stampa previsto, attribuitogli all'ingresso nel sistema

Suggerimenti

- **due code con priorità distinte** con gestione FIFO
- attenzione alla condizione di blocco nei due metodi di acquisizione
  - Necessario gestire la priorità
- attenzione alla uscita dalla coda:
  - escono gruppi di N_MAX thread, non più 1 thread alla volta
  - quale modifica dello stato considerare rilevante per sbloccare i thread in attesa?

Suggerimenti estensione

- **per ogni documento**, istante di stampa previsto va **calcolato all'ingresso** nella coda **ma stampato all'uscita** dalla coda
  - come tenere traccia di questa informazione man mano che l'esecuzione procede?