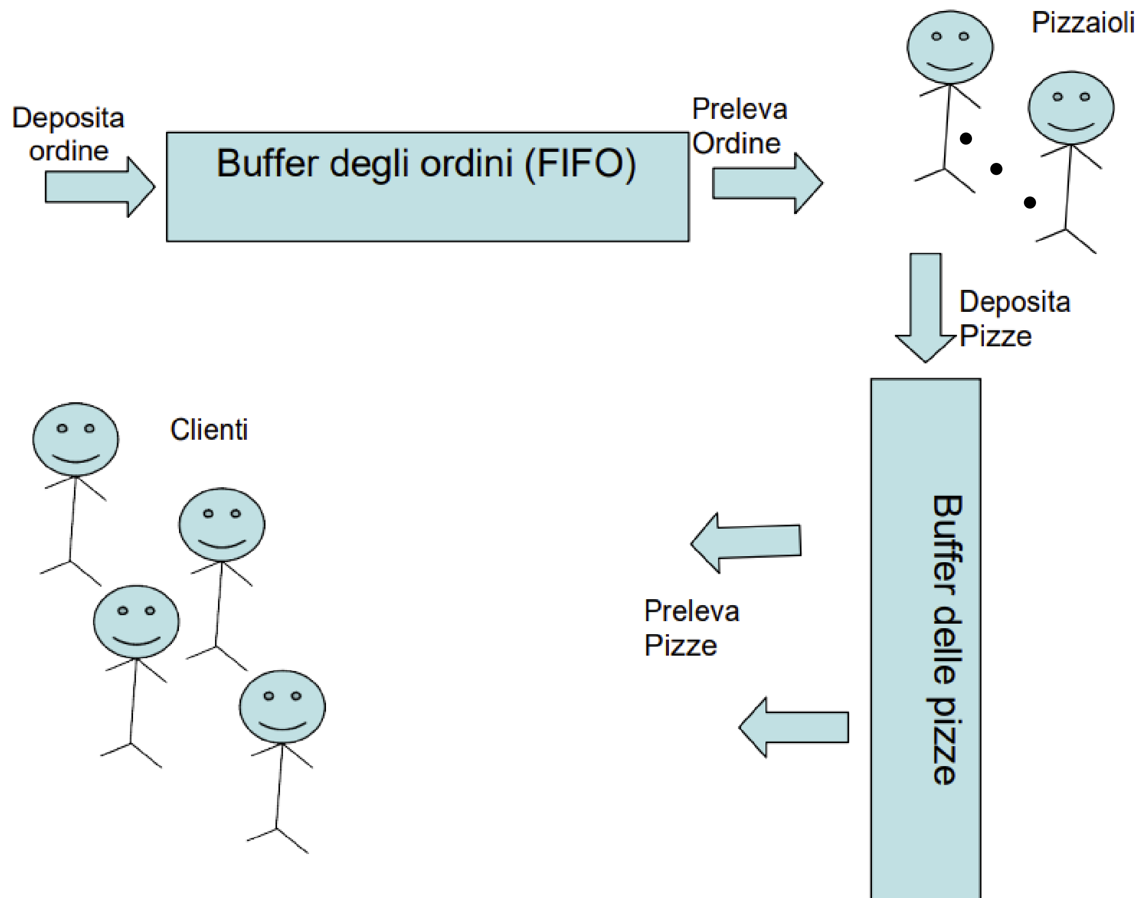
Estratto dall'esame di Sistemi Operativi del 18 Giugno 2007

La pizzeria "Da Totò" ha bisogno di un efficace metodo di smaltimento dei propri ordini secondo lo schema di figura.



La pizzeria funziona in base alla presenza di tre tipi di entità:

Pizzeria. Una pizzeria è' composta da due sotto-strutture dati:

1. Un buffer degli ordini (BO), di dimensione O, dove è possibile ordinare un certo quantitativo di pizze. Ogni ordine è composto da una coppia `<codicePizza, quantita>`, dove `codicePizza` indica il tipo di pizza (ad esempio 0=Margherita, 1=4 Stagioni, ecc.), mentre `quantita` indica il numero di pizze che sono state ordinate per ciascun tipo. In BO un cliente deposita un dato ordine con il metodo `putOrdine`, che restituisce un riferimento all'ordine formulato. `putOrdine` deve bloccare il thread chiamante se non ci sono slot disponibili in BO. E' possibile prelevare un ordine da BO con il metodo `getOrdine`, che restituisce l'ordine prelevato. La politica di gestione di BO è FIFO. `getOrdine` deve bloccare il thread chiamante finché non c'è un ordine disponibile in BO.

2. Un buffer delle pizze (BP), di dimensione P, dove è possibile depositare le pizze appena pronte. In ogni slot di questo buffer è possibile depositare un certo ordine dopo che questo è stato lavorato da un pizzaiolo (si suppone che in ogni slot possa essere sistemata una pila di cartoni contenenti tutte le pizze relative a un certo ordine). Le pizze pronte vengono depositate con il metodo `putPizze.` `putPizze` riceve in input un ordine. Il thread chiamante deve bloccarsi in attesa se non ci sono slot liberi in BP. Le pizze vengono prelevate dai clienti attraverso il metodo `getPizze` che riceve in input il riferimento a un certo ordine. Il metodo blocca il thread chiamante nel caso in cui l'ordine richiesto non sia stato ancora servito. Si noti che la politica di gestione di BP non deve essere necessariamente FIFO.

Cliente. Il cliente è un tipo di thread il cui ciclo di vita è il seguente:

a) Inattività. Il cliente non fa nulla di particolare;
b) Sottomissione ordine. Il cliente genera un ordine e lo sottopone alla pizzeria;
c) Attesa non bloccante. Il cliente esegue altri compiti prima di richiedere il proprio ordine; per simulare questo periodo in cui il Cliente non aspetta l'ordine (es. il cliente va a farsi un giro attorno alla pizzeria), si può far passare un certo periodo attendendo per un certo tempo scelto casualmente, ma ragionevolmente proporzionale al numero di pizze attese.
d) Attesa bloccante. Il cliente esegue `getPizze` ed eventualmente si mette in attesa che l'ordine con il proprio codice sia servito all'interno di BP;
e) Ritorno al punto a.

Pizzaiolo. Il pizzaiolo è un tipo di thread con il seguente ciclo di vita:

a) Prelievo di un ordine da BO. Si blocca nel caso non ci siano ordini disponibili in BO.
b) Elaborazione dell'ordine. Si simuli un tempo di elaborazione dell'ordine proporzionale al numero di pizze da preparare.
c) Erogazione dell'ordine in BP. Il pizzaiolo deposita le pizze in BP, o si blocca in attesa se in BP non ci sono slot.
d) Ritorno al punto a.

Si progettino le classi `Pizzeria`, `Cliente` e `Pizzaiolo`, nonché un programma `main` che avvii la pizzeria `Da_Toto`, con `C` clienti e `P` pizzaioli ad essa collegati.
La soluzione proposta dallo studente dovrebbe garantire (in rigoroso ordine di priorità)

1. Accesso in mutua esclusione alle strutture dati della pizzeria;
2. *Prevenzione di qualsiasi possibile situazione di deadlock;*
3. *Prevenzione di possibili situazioni di starvation;*
4. Utilizzo ottimale delle risorse (non è ad esempio naturale che BP sia gestito in politica FIFO: alcuni ordini potrebbero essere pronti non nell'ordine in cui sono stati formulati, causando attese inutili).

E' consentito il riuso del codice della classe `BlockingQueue` vista a lezione, opportunamente riadattata, di qualsiasi altra classe predefinita di Python 3.9 o successivo, incluse tutte le strutture dati presenti nel modulo `queue` .