Semplice problema **Prod/Cons** con un **singolo buffer condiviso**.  
L’utente decide quale thread creare scrivendolo in **input**, ovviamente il tutto all’interno di un ciclo while infinito.

In questo caso i **thread** **Producer** e **Consumer** sono stati definiti per **ereditarietà**, e lavoreranno su un **Buffer** comune.

I **Thread** semplicemente andranno a **sovrascrivere** il metodo **run**(), avviati poi da **start**() nel main, ed invocheranno poi metodi sull’oggetto buffer.

Avranno infatti un **dato membro** che è proprio il **buffer**, su cui invocheranno i metodi **produci**() o **consuma**().

Il **problema in questo caso** (con un unico buffer, un’unica **cella**) con **un produttore** e un **consumatore**: è un problema di **cooperazione**, non c’è competizione perché i vincoli imposti dal problema impediscono la competizione. Dato che devo **imporre un preciso ordine** di esecuzione delle operazione di **produzione e consumo** non ho competizione.

**Buffer** definisce i metodi **produci** e **consuma** **sincronyzed**.

* Quando il **buffer** è **pieno**, i **produttori** chiamano **wait() per rilasciare il lock** associato all’oggetto Buffer e attendere che il buffer sia vuoto.
* Quando il **buffer** è **vuoto**, i **consumatori** chiamano **wait() per rilasciare il lock** associato all’oggetto Buffer e attendere che il buffer sia pieno.
* Quando un produttore o un consumatore modifica lo stato del buffer, chiama notifyAll() per notificare tutti i thread in attesa che lo stato del buffer è cambiato.

Ricorda che i **while** in cui si trovano le **wait** è dovuto dal fatto che i **monitor** sono **wait\_and\_continue**.  
In più la **wait** viene effettuata su un’unica variabile condition implicita dei monitor java.