**OBBIETTIVO DEL PROGRAMMA EXAMPLE01:**

Costruiamo un programma con due thread che il primo parte e si mette in attesa, il secondo parte e fa un’attesa casuale, a questo punto notifica al primo thread che termina.

------------------------------------------------------------------------------------------

go(); metodo che possiamo mettere pubblico perché è un metodo alternativo per chiamare main.

In questo esercizio ci vuole uno stato in comune perché se il waiter deve fare wait su un oggetto e il notifier deve fare notify su un oggetto, devono essere lo stesso oggetto e quindi uno stato in comune, un oggetto che possiamo usare è Example01, perché se example01 è l’oggetto che crea waiter e notfier e li lancia, è il suo stato che quei due devono condividere;

**Perché dobbiamo gestire la sleep o le attese in generale?**

La sleep e le attese come la .wait devono essere gestite tramite try-catch dove nel try si mette l’attesa e come argomento del catch catturiamo Throwable throwable perché è un’eccezione che per java bisogna gestire, catturandola o farla uscire.

ES: try{

Thread.sleep(2000);

}catch(Throwable throwable){

//Blank

}

**Serve fare una classe esterna?**

Può sembra una soluzione ma non condivide lo stato.

**Come si può fare a rendere una classe talmente unita ad un’altra classe in modo da condividere lo stato della classe più esterna?**

Si mette la classe dentro alle parentesi graffe della classe contenitrice.

**Inner class(classe interna)**: ne possiamo mettere quante ne vogliamo, ogni classe interna vede lo stato della classe che la contiene, si possono fare anche anonime, che sono usate esclusivamente per fare la new degli oggetti, senza necessariamente darle un nome.

**L’oggetto mutex dove bisogna inserirlo?**

L’oggetto mutex ha bisogno di essere in una regione critica, se non è messo al suo interno verrà fuori l’errore “IllegalMonitorStateExcepion” , che si risolve con il blocco synchronized(**mutex**); importante deve avere lo stesso oggetto in questo caso **mutex.**

**Funzionalità di JAVA 8:** Possiamo è andare a costruire l’implementazione di un’interfaccia e la definizione del metodo dell’interfaccia tutti insieme.

ES: Thread notifier = new Thread(this::doNotify)

Facendo così andiamo a costruire una lambda expression che chiama doNotifiy sull’oggetto this.

**A cosa serve la variabile waitInProgress?**

Permette alla Notify di aspettare che venga eseguita la wait; quindi, mettere il cambio a true nella regione critica così da mantenere l’atomicità e che abbia una sua notify, nella Notify bisogna testarlo che appunto ci sia questo cambio, usiamo una while perché se qualcuno fuori fa notify e ci sblocca la wait noi andiamo a ricontrollare se waitInProgress abbia cambiato o meno stato così da capire se avanzare o meno.

Alla fine della go però lo stato del waitInProgress rimane true e quindi non più riutilizzabile allora, prima di far partire tutto la settiamo a false(ridonante ma più sicuro).

Tutte le volte che abbiamo un oggetto che il cui stato viene usato per la sincronizzazione dobbiamo essere sicuro di proteggerlo il più possibile, ciò di non farlo mai vedere fuori.