

4-La classe `CountDownLatch` permette di sincronizzare uno o più thread che devono attendere, senza consumare CPU, il completamento di operazioni in corso in altri thread. All'atto della costruzione, gli oggetti di questa classe contengono un contatore inizializzato con un valore intero strettamente positivo. Oltre al costruttore, questa classe offre due soli metodi pubblici: `void await()` e `void countDown()`. Quando un thread invoca `await()`, rimane bloccato fino a che il contatore non raggiunge il valore 0, dopodiché ritorna; se, viceversa, all'atto della chiamata il contatore vale già 0, il metodo ritorna immediatamente. Quando viene invocato `countDown()`, se il contatore è maggiore di zero, viene decrementato e se, come conseguenza del decremento, diventa nullo libera i thread bloccati all'interno di `await()`. Se, viceversa, il contatore valeva già a zero, l'invocazione di `countDown()` non ha effetti. Si implementi tale classe utilizzando le librerie standard C++11.(3.0)

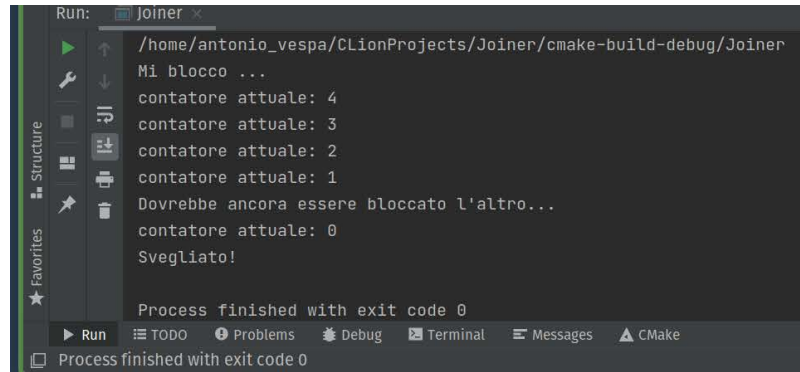
[Commenti dal prof:
Costruttore e metodi `await()` e `countDown()` devono essere pubblici.
Non hai verificato che il parametro passato al costruttore sia non negativo
]

```

1 #include <iostream>
2 #include <map>
3 #include <thread>
4 #include <condition_variable>
5 #include <mutex>
6 #include <chrono>
7 using namespace std::chrono_literals;
8
9 class CountDownLatch {
10 private:
11     int cont; //non rispecchia il numero di thread che usa questa classe
12     std::mutex m;
13     std::condition_variable cv;
14 public:
15     CountDownLatch(int n):cont(n){
16         if(n<=0)
17             throw std::logic_error("parametro deve essere > 0");
18     };
19
20     void await(){ //thread bloccato fin quando cont=0
21         std::unique_lock<std::mutex> ul(m);
22         std::cout << "Mi blocco ..." << std::endl;
23         cv.wait(ul,[this](){return (cont==0);});
24         std::cout << "Svegliato! " << std::endl;
25         //si sblocca e torna
26     };
27
28     void countDown(){ //cont--
29         std::unique_lock<std::mutex> ul(m);
30         if(cont==0)
31             return;
32         cont--;
33         std::cout << "contatore attuale: " << cont << std::endl;
34         if(cont==0)
35             cv.notify_all();
36     };
37
38 };
39
40 int main() {
41     CountDownLatch cl(5);
42
43     std::thread t1([&cl](){
44         //std::this_thread::sleep_for(500ms);
45         cl.await();
46     });
47
48     std::thread t2([&cl](){
49         std::this_thread::sleep_for(500ms);
50         cl.countDown();
51         cl.countDown();
52         cl.countDown();
53         cl.countDown();
54         std::cout << "Dovrebbe ancora essere bloccato l'altro..." << std::endl;
55     });
56
57     std::thread t3([&cl](){
58         std::this_thread::sleep_for(1500ms);
59         cl.countDown();
60     });

```

```
61  
62     t1.join();  
63     t2.join();  
64     t3.join();  
65     return 0;  
66 }  
67
```



```
Run: Joiner x  
/home/antonio-vespa/CLionProjects/Joiner/cmake-build-debug/Joiner  
Mi blocco ...  
contatore attuale: 4  
contatore attuale: 3  
contatore attuale: 2  
contatore attuale: 1  
Dovrebbe ancora essere bloccato l'altro...  
contatore attuale: 0  
Svegliato!  
Process finished with exit code 0
```