```
2)  import time
3)  l=Lock()
4)  def wish(name):
5)      l.acquire()
6)      for i in range(10):
7)          print("Good Evening:",end='')
8)          time.sleep(2)
9)          print(name)
10)     l.release()
11)
12) t1=Thread(target=wish,args=("Dhoni",))
13) t2=Thread(target=wish,args=("Yuvraj",))
14) t3=Thread(target=wish,args=("Kohli",))
15) t1.start()
16) t2.start()
17) t3.start()
```

In the above program at a time only one thread is allowed to execute wish() method and hence we will get regular output.

Problem with Simple Lock:

The standard Lock object does not care which thread is currently holding that lock.If the lock is held and any thread attempts to acquire lock, then it will be blocked,even the same thread is already holding that lock.

Eg:

```
1)  from threading import *
2)  l=Lock()
3)  print("Main Thread trying to acquire Lock")
4)  l.acquire()
5)  print("Main Thread trying to acquire Lock Again")
6)  l.acquire()
```

Output:

```
D:\python_classes>py test.py
Main Thread trying to acquire Lock
Main Thread trying to acquire Lock Again
--
```

In the above Program main thread will be blocked b'z it is trying to acquire the lock second time.

Note: To kill the blocking thread from windows command prompt we have to use ctrl+break. Here ctrl+C won't work.

If the Thread calls recursive functions or nested access to resources,then the thread may trying to acquire the same lock again and again,which may block our thread.