Java : ReEnterantLock

**Author : Mradulanand Sharma**

Version History

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Modified By | Description | Version |
| Jul 24, 2012 | Mradulanand Sharma | First Draft | T1.1 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Contents

[1.> Introduction of another mechanism of thread synchronization in Java : java.util.concurrent.locks.Lock 4](#_Toc109589778)

[2.> High level understanding of classes and their association: class diagram. 6](#_Toc109589779)

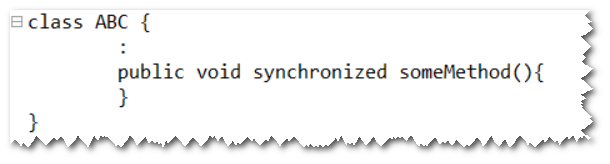
[3.> Execution – 01: A2 checked if lock is acquired, if yes then sleep for some time else continue later. 8](#_Toc109589780)

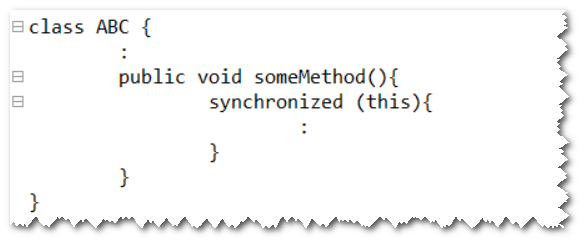
[4.> Execution – 02: A2 did not check if lock is already acquired, but it did take lock on shared object before continuing. 9](#_Toc109589781)

[5.> Execution – 03: A2 neither check if lock is already acquired, nor it requested for lock, but it tries to modify other variable than what thread A1 modifies. 11](#_Toc109589782)

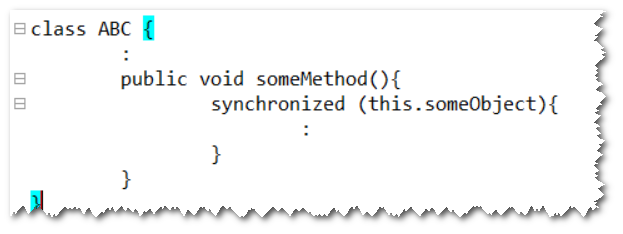
# Introduction of another mechanism of thread synchronization in Java : java.util.concurrent.locks.Lock

* We can create method synchronized, but that takes lock on entire object where method is marked synchronized. Example as follows, both are pretty same:

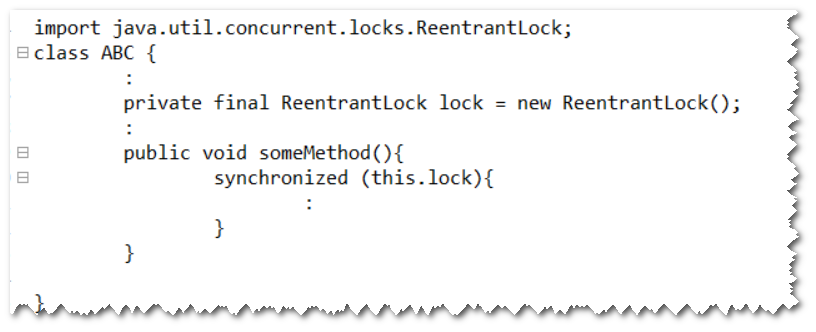




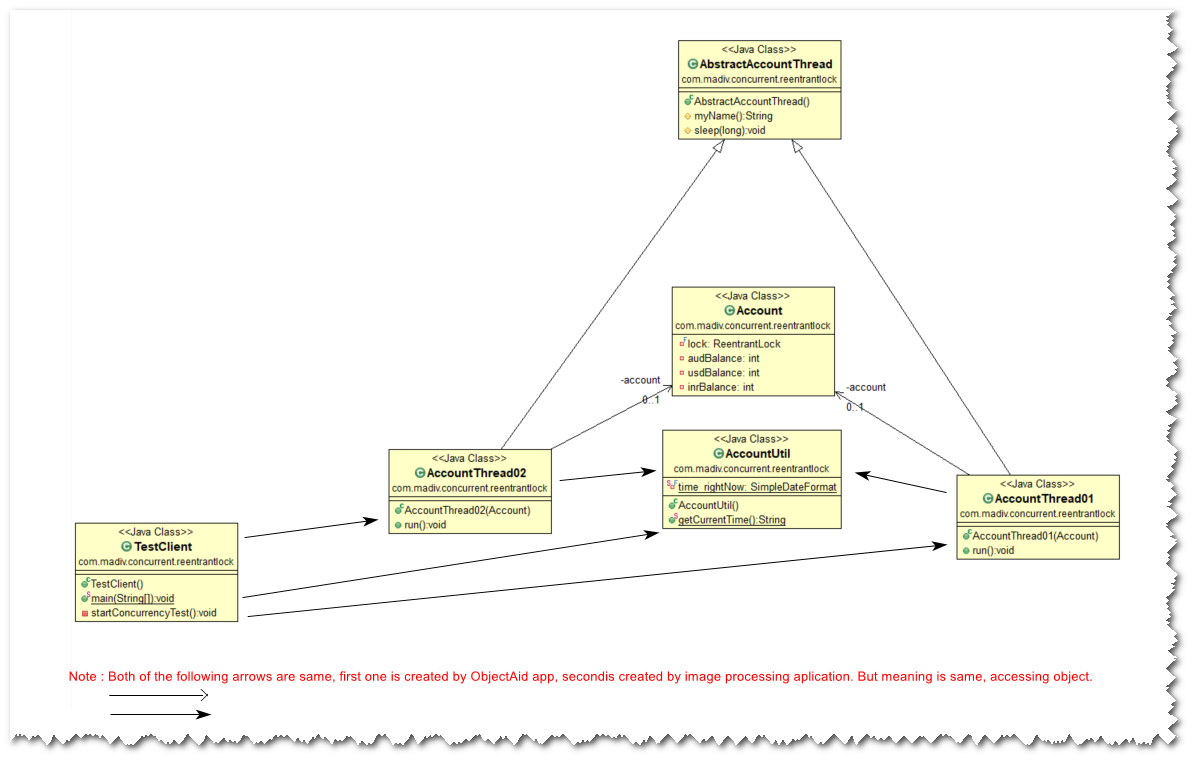
* Or we can synchronize on ABC class some specific object or some another object, that way any thread will not take lock on entire object of class ABC but part of its another member variable. Threads can easily enter into methods and as soon as some thread is trying to access some object subject to be synchronized, rest of the threads will be blocked. This is better of synchronizing thread instead of method level synchronization or synchronizing on this object.



* Synchronized block cannot be created on primitive variables hence as per java specification it is good practice to create object of ***java.util.concurrent.locks.Lock*** and every thread then can take lock of that Lock object. One of such lock implementation given by java is ***java.util.concurrent.locks.ReentrantLock*** following is one of such example.



# High level understanding of classes and their association: class diagram.



***Acount*** class is shared resource. ***AccountThread01*** and ***AccountThread02*** are two threads working on shared resource ***Account***. Common methods which both thread want, is moved to abstract class ***AbstractAccountThread*** and both thread is extending this abstract class. ***AccountUtil*** is some utility class used by both threads and client, just for some logging purpose.

Account Class: AbstractAccountThread:

AccountThread01: AccountThread02:

AccountUtil:



Client:



# Execution – 01: A2 checked if lock is acquired, if yes then sleep for some time else continue later.

Same set of classes attached in above section, where A2 thread is checking if lock is already acquired, if yes then sleep for some time else continue. When client is executed, following is the output:

2022-07-24 19:05:13.870 : account before = Account [audBalance=0, usdBalance=0, inrBalance=0]

2022-07-24 19:05:13.870 : A1 : Started...

2022-07-24 19:05:13.870 : A1 : Aquired lock, and sleeping for 5000ms

2022-07-24 19:05:14.073 : A2 : Started...

2022-07-24 19:05:14.073 : A2 : account.getUsdBalance() = 0

2022-07-24 19:05:14.073 : A2 : Lock found on accout object, sleeping for 1000ms.

2022-07-24 19:05:15.088 : A2 : Lock found on accout object, sleeping for 1000ms.

2022-07-24 19:05:16.104 : A2 : Lock found on accout object, sleeping for 1000ms.

2022-07-24 19:05:17.104 : A2 : Lock found on accout object, sleeping for 1000ms.

2022-07-24 19:05:18.120 : A2 : Lock found on accout object, sleeping for 1000ms.

2022-07-24 19:05:18.870 : A1 : Fin

2022-07-24 19:05:19.135 : A2 : Aquiring lock and continue.

2022-07-24 19:05:19.135 : A2 : Fin

2022-07-24 19:05:19.135 : account after = Account [audBalance=150, usdBalance=250, inrBalance=350]

2022-07-24 19:05:19.136 : Fin !!

# Execution – 02: A2 did not check if lock is already acquired, but it did take lock on shared object before continuing.

Same set of classes attached in above section, only class AccountThread02.java is changed a bit, attaching below:



2022-07-24 19:17:23.385 : account before = Account [audBalance=0, usdBalance=0, inrBalance=0]

2022-07-24 19:17:23.385 : A1 : Started...

2022-07-24 19:17:23.385 : A1 : Aquired lock, and sleeping for 5000ms

2022-07-24 19:17:23.591 : A2 : Started...

2022-07-24 19:17:23.591 : A2 : account.getUsdBalance() = 0

2022-07-24 19:17:23.591 : A2 : Aquiring lock and continue.

2022-07-24 19:17:28.385 : A1 : Fin

2022-07-24 19:17:28.385 : A2 : Fin

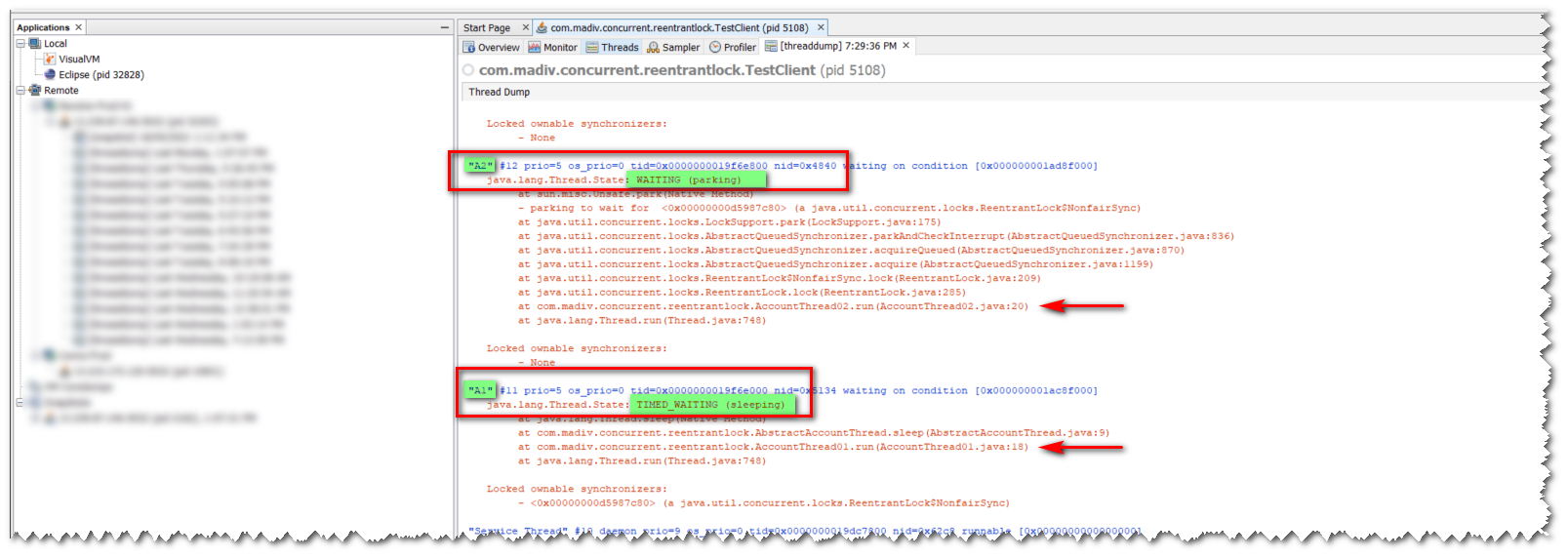
2022-07-24 19:17:28.386 : account after = Account [audBalance=150, usdBalance=250, inrBalance=350]

2022-07-24 19:17:28.386 : Fin !!

***jvisualvm*** was started and thread dump was taken (with perfect timing). Following was output of thread dump, it is clear that thread A1 is time waiting (sleeping for 5 seconds) and second thread A2 is just waiting for that resource accessibility.

TIMED\_WAITING thread: Thread which is waiting for some defined time, and then will continue when that time is over.

WAITING thread: Thread which is waiting for resource to get free, do not know when thread will be free.



# Execution – 03: A2 neither check if lock is already acquired, nor it requested for lock, but it tries to modify other variable than what thread A1 modifies.

Same set of classes attached in above section, only class AccountThread02.java is changed a bit, attaching below:



Execution log: We can see, thread A2 does not wait for thread A1, it continues and does what it is supposed to do.

2022-07-24 19:45:01.276 : account before = Account [audBalance=0, usdBalance=0, inrBalance=0]

2022-07-24 19:45:01.291 : A1 : Started...

2022-07-24 19:45:01.291 : A1 : Aquired lock, and sleeping for 5000ms

2022-07-24 19:45:01.494 : A2 : Started...

2022-07-24 19:45:01.494 : A2 : account.getUsdBalance() = 0

2022-07-24 19:45:01.494 : A2 : Fin

2022-07-24 19:45:06.291 : A1 : Fin

2022-07-24 19:45:06.291 : account after = Account [audBalance=150, usdBalance=250, inrBalance=350]

2022-07-24 19:45:06.291 : Fin !!