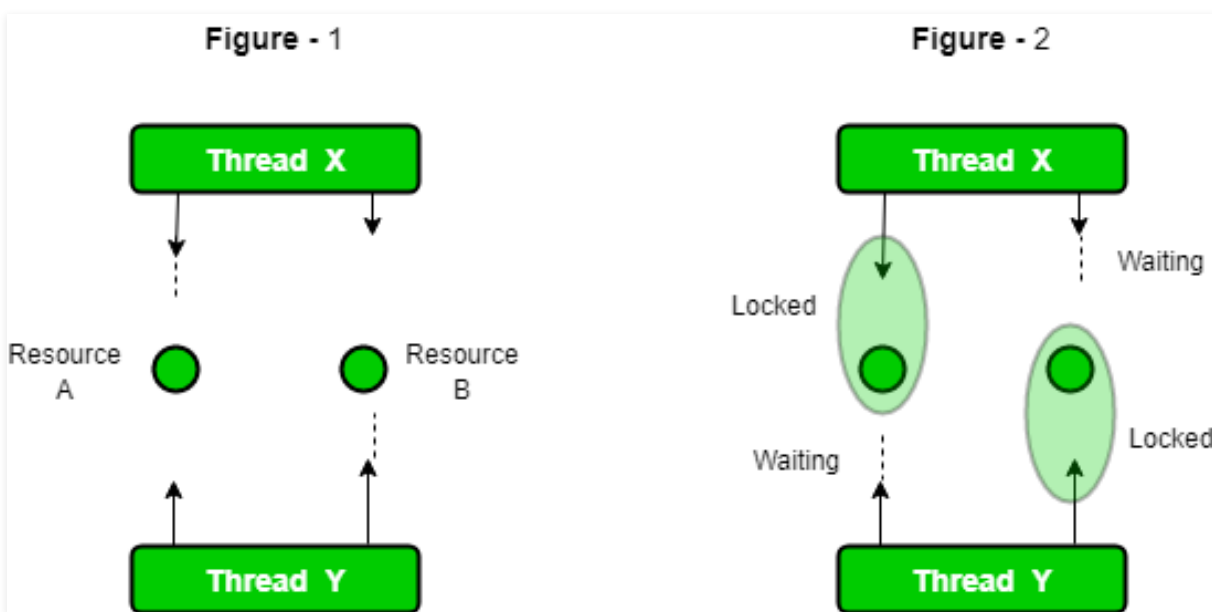


# Deadlock in Java Multithreading

Difficulty Level : Expert Last Updated : 28 Jun, 2021

**synchronized** keyword is used to make the class or method thread-safe which means only one thread can have lock of synchronized method and use it, other threads have to wait till the lock releases and anyone of them acquire that lock.

It is important to use if our program is running in multi-threaded environment where two or more threads execute simultaneously. But sometimes it also causes a problem which is called **Deadlock**. Below is a simple example of Deadlock condition.



```
// Java program to illustrate Deadlock
// in multithreading.
class Util
{
    // Util class to sleep a thread
    static void sleep(long millis)
    {
        try
        {
            Thread.sleep(millis);
        }
        catch (InterruptedException e)
        {
        }
    }
}
```

```

        e.printStackTrace();
    }
}

// This class is shared by both threads
class Shared
{
    // first synchronized method
    synchronized void test1(Shared s2)
    {
        System.out.println("test1-begin");
        Util.sleep(1000);

        // taking object lock of s2 enters
        // into test2 method
        s2.test2();
        System.out.println("test1-end");
    }

    // second synchronized method
    synchronized void test2()
    {
        System.out.println("test2-begin");
        Util.sleep(1000);
        // taking object lock of s1 enters
        // into test1 method
        System.out.println("test2-end");
    }
}

```

```

class Thread1 extends Thread
{
    private Shared s1;
    private Shared s2;

    // constructor to initialize fields
    public Thread1(Shared s1, Shared s2)
    {
        this.s1 = s1;
        this.s2 = s2;
    }

    // run method to start a thread
    @Override
    public void run()
    {
        // taking object lock of s1 enters
        // into test1 method
        s1.test1(s2);
    }
}

```

```

class Thread2 extends Thread

```

```
{
    private Shared s1;
    private Shared s2;

    // constructor to initialize fields
    public Thread2(Shared s1, Shared s2)
    {
        this.s1 = s1;
        this.s2 = s2;
    }

    // run method to start a thread
    @Override
    public void run()
    {
        // taking object lock of s2
        // enters into test2 method
        s2.test1(s1);
    }
}

public class Deadlock
{
    public static void main(String[] args)
    {
        // creating one object
        Shared s1 = new Shared();

        // creating second object
        Shared s2 = new Shared();

        // creating first thread and starting it
        Thread1 t1 = new Thread1(s1, s2);
        t1.start();

        // creating second thread and starting it
        Thread2 t2 = new Thread2(s1, s2);
        t2.start();

        // sleeping main thread
        Util.sleep(2000);
    }
}
```

Output : test1-begin  
test2-begin

It is not recommended to run the above program with online IDE. We can copy the source code and run it on our local machine. We can see that it runs for indefinite time, because threads are in deadlock condition and doesn't let code to execute. Now let's see step by step what is happening there.

1. Thread t1 starts and calls test1 method by taking the object lock of s1.
2. Thread t2 starts and calls test1 method by taking the object lock of s2.
3. t1 prints test1-begin and t2 prints test-2 begin and both waits for 1 second, so that both threads can be started if any of them is not.
4. t1 tries to take object lock of s2 and call method test2 but as it is already acquired by t2 so it waits till it become free. It will not release lock of s1 until it gets lock of s2.
5. Same happens with t2. It tries to take object lock of s1 and call method test1 but it is already acquired by t1, so it has to wait till t1 release the lock. t2 will also not release lock of s2 until it gets lock of s1.
6. Now, both threads are in wait state, waiting for each other to release locks. Now there is a race around condition that who will release the lock first.
7. As none of them is ready to release lock, so this is the Dead Lock condition.
8. When you will run this program, it will be look like execution is paused.

## Detect Dead Lock condition

We can also detect deadlock by running this program on cmd. We have to collect Thread Dump. Command to collect depends on OS type. If we are using Windows and Java 8, command is `jcmd $PID Thread.print`

We can get PID by running `jps` command. Thread dump for above program is below:

```
jcmd 18692 Thread.print
18692:
2020-06-08 19:03:10
Full thread dump OpenJDK 64-Bit Server VM (11.0.4+10-b304.69 mixed mode, shar

Threads class SMR info:
_java_thread_list=0x0000017f44b69f20, length=13, elements={
0x0000017f43f77000, 0x0000017f43f79800, 0x0000017f43f90000, 0x0000017f43f9100
0x0000017f43f95000, 0x0000017f43fa5000, 0x0000017f43fb0800, 0x0000017f43f5b80
0x0000017f44bc9000, 0x0000017f44afb000, 0x0000017f44bd7800, 0x0000017f44bd880
```

```
0x0000017f298c9000
```

```
}
```

```
"Reference Handler" #2 daemon prio=10 os_prio=2 cpu=0.00ms elapsed=57.48s tid
  java.lang.Thread.State: RUNNABLE
```

```
    at java.lang.ref.Reference.waitForReferencePendingList(java.base@11.0
    at java.lang.ref.Reference.processPendingReferences(java.base@11.0.4/
    at java.lang.ref.Reference$ReferenceHandler.run(java.base@11.0.4/Refe
```

```
"Finalizer" #3 daemon prio=8 os_prio=1 cpu=0.00ms elapsed=57.48s tid=0x000001
  java.lang.Thread.State: WAITING (on object monitor)
```

```
    at java.lang.Object.wait(java.base@11.0.4/Native Method)
    - waiting on (a java.lang.ref.ReferenceQueue$Lock)
    at java.lang.ref.ReferenceQueue.remove(java.base@11.0.4/ReferenceQueu

    - waiting to re-lock in wait() (a java.lang.ref.ReferenceQueue$Lock)
    at java.lang.ref.ReferenceQueue.remove(java.base@11.0.4/ReferenceQueu
    at java.lang.ref.Finalizer$FinalizerThread.run(java.base@11.0.4/Final
```

```
"Signal Dispatcher" #4 daemon prio=9 os_prio=2 cpu=0.00ms elapsed=57.47s tid=
  java.lang.Thread.State: RUNNABLE
```

```
"Attach Listener" #5 daemon prio=5 os_prio=2 cpu=31.25ms elapsed=57.47s tid=0
  java.lang.Thread.State: RUNNABLE
```

```
"C2 CompilerThread0" #6 daemon prio=9 os_prio=2 cpu=46.88ms elapsed=57.47s ti
  java.lang.Thread.State: RUNNABLE
```

```
    No compile task
```

```
"C1 CompilerThread0" #9 daemon prio=9 os_prio=2 cpu=93.75ms elapsed=57.47s ti
  java.lang.Thread.State: RUNNABLE
```

```
    No compile task
```

```
"Sweeper thread" #10 daemon prio=9 os_prio=2 cpu=0.00ms elapsed=57.47s tid=0x
  java.lang.Thread.State: RUNNABLE
```

```
"Common-Cleaner" #11 daemon prio=8 os_prio=1 cpu=0.00ms elapsed=57.44s tid=0x
  java.lang.Thread.State: TIMED_WAITING (on object monitor)
```

```
    at java.lang.Object.wait(java.base@11.0.4/Native Method)
```

```

- waiting on (a java.lang.ref.ReferenceQueue$Lock)
at java.lang.ref.ReferenceQueue.remove(java.base@11.0.4/ReferenceQueue
- waiting to re-lock in wait() (a java.lang.ref.ReferenceQueue$Lock)
at jdk.internal.ref.CleanerImpl.run(java.base@11.0.4/CleanerImpl.java
at java.lang.Thread.run(java.base@11.0.4/Thread.java:834)
at jdk.internal.misc.InnocuousThread.run(java.base@11.0.4/InnocuousTh

```

"Monitor Ctrl-Break" #12 daemon prio=5 os\_prio=0 cpu=15.63ms elapsed=57.36s t  
 java.lang.Thread.State: RUNNABLE

```

at java.net.SocketInputStream.socketRead0(java.base@11.0.4/Native Met
at java.net.SocketInputStream.socketRead(java.base@11.0.4/SocketInput
at java.net.SocketInputStream.read(java.base@11.0.4/SocketInputStream
at java.net.SocketInputStream.read(java.base@11.0.4/SocketInputStream
at sun.nio.cs.StreamDecoder.readBytes(java.base@11.0.4/StreamDecoder.
at sun.nio.cs.StreamDecoder.implRead(java.base@11.0.4/StreamDecoder.j
at sun.nio.cs.StreamDecoder.read(java.base@11.0.4/StreamDecoder.java:
- locked (a java.io.InputStreamReader)
at java.io.InputStreamReader.read(java.base@11.0.4/InputStreamReader.
at java.io.BufferedReader.fill(java.base@11.0.4/BufferedReader.java:1
at java.io.BufferedReader.readLine(java.base@11.0.4/BufferedReader.ja
- locked (a java.io.InputStreamReader)
at java.io.BufferedReader.readLine(java.base@11.0.4/BufferedReader.ja
at com.intellij.rt.execution.application.AppMainV2$1.run(AppMainV2.ja

```

"Service Thread" #13 daemon prio=9 os\_prio=0 cpu=0.00ms elapsed=57.36s tid=0x  
 java.lang.Thread.State: RUNNABLE

"Thread-0" #14 prio=5 os\_prio=0 cpu=0.00ms elapsed=57.35s tid=0x0000017f44bd7  
 java.lang.Thread.State: BLOCKED (on object monitor)

```

at com.company.threads.Shared.test2(Deadlock.java:40)
- waiting to lock (a com.company.threads.Shared)
at com.company.threads.Shared.test1(Deadlock.java:33)
- locked (a com.company.threads.Shared)
at com.company.threads.Thread1.run(Deadlock.java:67)

```

"Thread-1" #15 prio=5 os\_prio=0 cpu=0.00ms elapsed=57.35s tid=0x0000017f44bd8  
 java.lang.Thread.State: BLOCKED (on object monitor)

```

at com.company.threads.Shared.test2(Deadlock.java:40)
- waiting to lock (a com.company.threads.Shared)

```

```

at com.company.threads.Shared.test1(Deadlock.java:33)
- locked (a com.company.threads.Shared)
at com.company.threads.Thread2.run(Deadlock.java:90)

```

```

"DestroyJavaVM" #16 prio=5 os_prio=0 cpu=171.88ms elapsed=55.35s tid=0x000001
  java.lang.Thread.State: RUNNABLE

```

```

"VM Thread" os_prio=2 cpu=0.00ms elapsed=57.49s tid=0x0000017f43f73800 nid=0x

```

```

"GC Thread#0" os_prio=2 cpu=0.00ms elapsed=57.51s tid=0x0000017f298e1000 nid=

```

```

"G1 Main Marker" os_prio=2 cpu=0.00ms elapsed=57.51s tid=0x0000017f29911000 n

```

```

"G1 Conc#0" os_prio=2 cpu=0.00ms elapsed=57.51s tid=0x0000017f29912000 nid=0x

```

```

"G1 Refine#0" os_prio=2 cpu=0.00ms elapsed=57.50s tid=0x0000017f43e0a800 nid=

```

```

"G1 Young RemSet Sampling" os_prio=2 cpu=0.00ms elapsed=57.50s tid=0x0000017f

```

```

"VM Periodic Task Thread" os_prio=2 cpu=0.00ms elapsed=57.36s tid=0x0000017f4

```

```

JNI global refs: 15, weak refs: 0

```

```

Found one Java-level deadlock:

```

```

=====

```

```

"Thread-0":

```

```

  waiting to lock monitor 0x0000017f43f87980 (object 0x000000008a2e9ce0, a co
  which is held by "Thread-1"

```

```

"Thread-1":

```

```

  waiting to lock monitor 0x0000017f43f87780 (object 0x000000008a2e9cd0, a co
  which is held by "Thread-0"

```

```

Java stack information for the threads listed above:

```

```

=====

```

```

"Thread-0":

```

```

  at com.company.threads.Shared.test2(Deadlock.java:40)
  - waiting to lock (a com.company.threads.Shared)
  at com.company.threads.Shared.test1(Deadlock.java:33)
  - locked (a com.company.threads.Shared)

```

```
at com.company.threads.Ihread1.run(Deadlock.java:67)
"Thread-1":
at com.company.threads.Shared.test2(Deadlock.java:40)
- waiting to lock (a com.company.threads.Shared)
at com.company.threads.Shared.test1(Deadlock.java:33)
- locked (a com.company.threads.Shared)
at com.company.threads.Thread2.run(Deadlock.java:90)
```

Found 1 deadlock.



As we can see there is clearly mentioned that found 1 deadlock. It is possible that the same message appears when you try on your machine.

## Avoid Dead Lock condition

We can avoid dead lock condition by knowing its possibilities. It's a very complex process and not easy to catch. But still if we try, we can avoid this. There are some methods by which we can avoid this condition. We can't completely remove its possibility but we can reduce.

- **Avoid Nested Locks** : This is the main reason for dead lock. Dead Lock mainly happens when we give locks to multiple threads. Avoid giving lock to multiple threads if we already have given to one.
- **Avoid Unnecessary Locks** : We should have lock only those members which are required. Having lock on unnecessarily can lead to dead lock.
- **Using thread join** : Dead lock condition appears when one thread is waiting other to finish. If this condition occurs we can use Thread.join with maximum time you think the execution will take.

## Important Points :

- If threads are waiting for each other to finish, then the condition is known as Deadlock.
- Deadlock condition is a complex condition which occurs only in case of multiple threads.
- Deadlock condition can break our code at run time and can destroy business logic.
- We should avoid this condition as much as we can.

This article is contributed by **Vishal Garg**. If you like GeeksforGeeks and would like to contribute, you can also write an article using [write.geeksforgeeks.org](https://www.geeksforgeeks.org/write/geeksforgeeks.org) or mail your article to



review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.