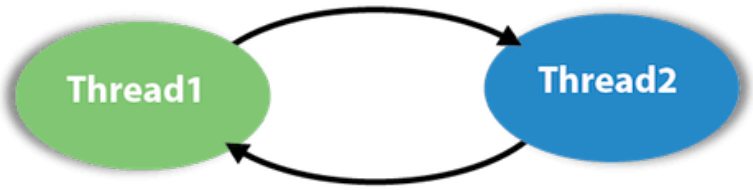
Deadlock in Java

Deadlock in Java is a part of multithreading. Deadlock can occur in a situation when a thread is waiting for an object lock, that is acquired by another thread and second thread is waiting for an object lock that is acquired by first thread. Since, both threads are waiting for each other to release the lock, the condition is called deadlock.



Example of Deadlock in Java

TestDeadlockExample1.java

```
public class TestDeadlockExample1 {
 public static void main(String[] args) {
  final String resource1 = "ratan jaiswal";
  final String resource2 = "vimal jaiswal";
  // t1 tries to lock resource1 then resource2
  Thread t1 = new Thread() {
   public void run() {
     synchronized (resource1) {
      System.out.println("Thread 1: locked resource 1");

      try { Thread.sleep(100);} catch (Exception e) {}

      synchronized (resource2) {
       System.out.println("Thread 1: locked resource 2");
      }
     }
   }
  };

  // t2 tries to lock resource2 then resource1
  Thread t2 = new Thread() {
   public void run() {
    synchronized (resource2) {
     System.out.println("Thread 2: locked resource 2");

     try { Thread.sleep(100);} catch (Exception e) {}

     (resource1) {
```

⇧ SCROLL TO TOP

```
            System.out.println("Thread 2: locked resource 1");
          }
        }
      }
    };



    t1.start();
    t2.start();
  }
}
```

Output:

```
Thread 1: locked resource 1
        Thread 2: locked resource 2
```

More Complicated Deadlocks

A deadlock may also include more than two threads. The reason is that it can be difficult to detect a deadlock. Here is an example in which four threads have deadlocked:

Thread 1 locks A, waits for B

Thread 2 locks B, waits for C

Thread 3 locks C, waits for D

Thread 4 locks D, waits for A

Thread 1 waits for thread 2, thread 2 waits for thread 3, thread 3 waits for thread 4, and thread 4 waits for thread 1.

How to avoid deadlock?

A solution for a problem is found at its roots. In deadlock it is the pattern of accessing the resources A and B, is the the issue we will have to simply re-order the statements where the code is accessing shared

⇧ SCROLL TO TOP

DeadlockSolved.java

```
public class DeadlockSolved {

   public static void main(String ar[]) {
      DeadlockSolved test = new DeadlockSolved();

      final resource1 a = test.new resource1();
      final resource2 b = test.new resource2();

  // Thread-1
Runnable b1 = new Runnable() {
   public void run() {
      synchronized (b) {
         try {
            /* Adding delay so that both threads can start trying to lock resources */
            Thread.sleep(100);
         } catch (InterruptedException e) {
            e.printStackTrace();
         }
         // Thread-1 have resource1 but need resource2 also
         synchronized (a) {
            System.out.println("In block 1");
         }
      }
   }
};

// Thread-2
Runnable b2 = new Runnable() {
   public void run() {
      synchronized (b) {
         // Thread-2 have resource2 but need resource1 also
         synchronized (a) {
            System.out.println("In block 2");
         }
      }
   }
};



      new Thread(b1).start();
                    2).start();
```

⇧ SCROLL TO TOP

```
    // resource1
    private class resource1 {
        private int i = 10;

        public int getI() {
            return i;
        }

        public void setI(int i) {
            this.i = i;
        }
    }

    // resource2
    private class resource2 {
        private int i = 20;

        public int getI() {
            return i;
        }

        public void setI(int i) {
            this.i = i;
        }
    }
}
```

Output:

```
In block 1
In block 2
```

⇧ SCROLL TO TOP

ass DeadlockSolved solves the deadlock kind of situation. It will help in avoiding deadlocks, and olving them.

How to Avoid Deadlock in Java?

Deadlocks cannot be completely resolved. But we can avoid them by following basic rules mentioned below:

1. **Avoid Nested Locks**: We must avoid giving locks to multiple threads, this is the main reason for a deadlock condition. It normally happens when you give locks to multiple threads.
2. **Avoid Unnecessary Locks**: The locks should be given to the important threads. Giving locks to the unnecessary threads that cause the deadlock condition.
3. **Using Thread Join**: A deadlock usually happens when one thread is waiting for the other to finish. In this case, we can use **join** with a maximum time that a thread will take.

← Prev Next →