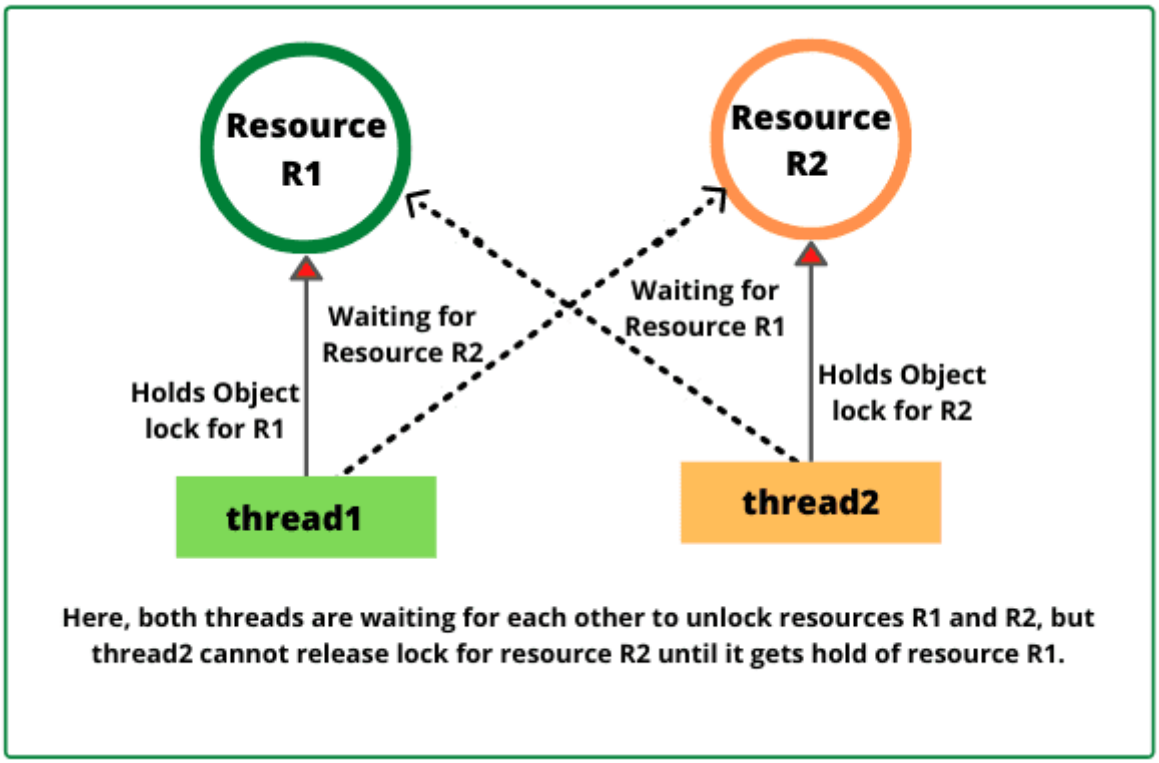
Deadlock in Java

Deadlock in Java is a situation that occurs 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 the first thread.

Since both threads are waiting for each other to release the lock simultaneously, this condition is called deadlock in Java. The object locks acquired by both threads are not released until their execution is not completed.




```
public class X
{
  void display1(X obj2)
  {
   System.out.println("Thread1 waiting for thread2 to release lock");
   synchronized(obj2) {
    System.out.println("Deadlock occurred");
   }
   }
```

```
void display2(X obj1)
{
 System.out.println("Thread2 waiting for thread1 to release lock");
 synchronized(obj1){
   System.out.println("Deadlock occurred");
  }
 }


}
```

```
public class Thread1 extends Thread
{
 X obj1, obj2;
   Thread1(X obj1, X obj2)
   {
     this.obj1 = obj1;
     this.obj2 = obj2;
   }
public void run()
{
   synchronized(obj1){
    try {
    Thread.sleep(1000);
    }
  catch(InterruptedException ie) {
   System.out.println(ie);
  }
obj2.display2(obj2);
  }
}
}
```

```
public class Thread2 extends Thread
{
  X obj1, obj2;
  Thread2(X obj1, X obj2)
  {
    this.obj1 = obj1;
    this.obj2 = obj2;
  }
  public void run()
  {
    synchronized(obj2){
    try {
```

```
        Thread.sleep(1000);
     }
    catch(InterruptedException ie) {
         System.out.println(ie);
    }
    obj1.display1(obj1);
 }
}
}
```

```
public class Deadlock {
public static void main(String[] args)
{
  X obj1 = new X();
  X obj2 = new X();

  Thread1 t1 = new Thread1(obj1, obj2);
  Thread2 t2 = new Thread2(obj1, obj2);
    t1.start();
    t2.start();
 }
}
```

What are the conditions for deadlock in a program?

Deadlock in a program may occur due to the following conditions. They are as follows:

1. Mutual Exclusion:
There is at least one resource that must be held in a non-sharable mode and hence, can be used only by one thread. If another thread requests for it, it should wait until the resource is available.

2. Hold and Wait: This condition occurs when one thread holds a resource and waits for another resource that is held by another thread.

3. No Preemption: The resource will be released only after the execution of thread is completed.

4. Circular Wait: This condition occurs when each thread is waiting for a resource held by the preceding one and the last thread is waiting for a resource held by first thread.This is called circular wait deadlock. This is because every thread is waiting for a resource held by next one and the last thread is waiting for a resource held by first.

How to avoid Deadlock in Java Program?

There is no
precise solution to overcome the problem of deadlock in a program. It depends on the logic used by the programmer. The programmer should create his program in such a way that it does not form the problem of deadlock.

A deadlock in a program can be prevented if any of the four conditions are not met. They are:

1. Mutual Exclusive Condition: If every resource is shared by multiple threads, deadlock would never occur.

2. Hold and Wait Condition: This condition can be eliminated when a thread is prohibited to wait
for more resources while already holding a certain resource. It can be achieved when we declare all resources at the very beginning that are expected to use by a thread.

3. No Preemption Condition: This condition can be eliminated if a thread holding a certain resource
is denied for further request. That thread must unlock its original resource. If necessary request them again together with additional resource.

4. Circular Wait Condition: This is the easiest way to avoid deadlock than the above three. There are two ways to eliminate deadlock.

Synchronized Block

Synchronized block in Java is another way of managing the execution of threads. It is mainly used to perform
synchronization on a certain block of code or statements inside the method.

Synchronizing a block of code is more powerful than *synchronized method*.

```
public class Table
{
    void printTable(int x)
   {
      synchronized(this) // Synchronized block.
     {
      for(int i = 1; i <= 3; i++)
      {
```

```
          System.out.println(x * i);
         try
        {
            Thread.sleep(400);
        }
       catch(InterruptedException ie)
      {
          System.out.println(ie);
      }
    }
   }
   }
}
public class Thread1 extends Thread
{
  Table t;
  Thread1(Table t)
 {
    this.t = t;
  }
public void run()
{
     t.printTable(2);
 }
}
public class Thread2 extends Thread
{
  Table t;
  Thread2(Table t)
  {
      this.t = t;
   }
public void run()
{
    t.printTable(10);
 }
}
public class SynchronizedBlock
{
public static void main(String[] args)
{
   Table t = new Table();
   Thread1 t1 = new Thread1(t);
   Thread2 t2 = new Thread2(t);
     t1.start();
     t2.start();
   }
}
```