ECAP615

Programming in Java

Harjinder Kaur

Assistant Professor

Learning Outcomes

After this lecture, you will be able to

- learn the basic concept Deadlock.
- understand the various methods of stopping a thread.
- implementation of deadlock situation and its solution.

Deadlock

- Deadlock describes a situation where two or more threads are blocked forever, waiting for each other.
- A Java multithreaded program may suffer from the deadlock condition because the synchronized keyword causes the executing thread to block while waiting for the lock, or monitor, associated with the specified object.

Deadlock

- 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.

Example

Figure - 1

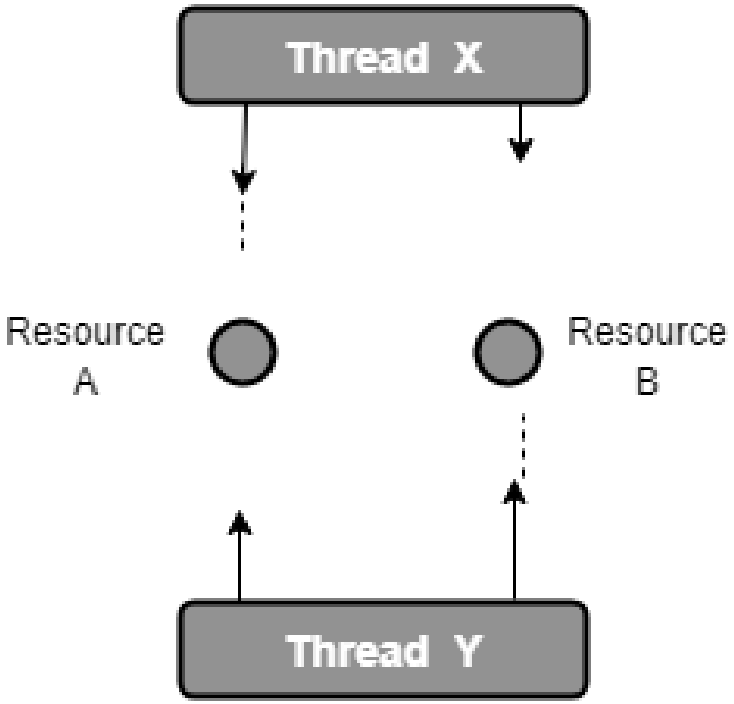
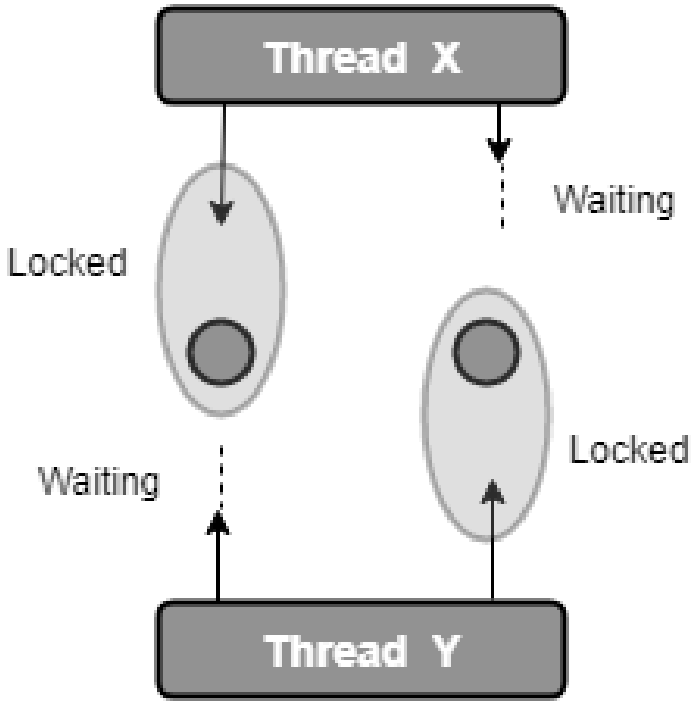


Figure - 2



Program Implementing Deadlock

```
public class Deadlock {
   public static Object Lock1 = new
Object();
   public static Object Lock2 = new
Object();
   public static void main
(String args[]) {
      ThreadD T1 = new ThreadD();
      ThreadD2 T2 = new ThreadD2();
      T1.start();
      T2.start();
  } } }
private static class ThreadD extends
Thread {
      public void run() {
         synchronized (Lock1) {
 System.out.println("Thread 1: Holding
lock 1...");
```

Continue....

Program Implementing Deadlock

Continue.

```
try { Thread.sleep(10); }
catch (InterruptedException e) {}
System.out.println("Thread 1: Waiting
for lock 2...");
synchronized (Lock2) {
        System.out.println("Thread 1:
Holding lock 1 & 2...");
        }
    }
  }
private static class ThreadD2
    extends Thread {
public void run() {
 synchronized (Lock2) {
   System.out.println("Thread 2:
    Holding lock 2...");
```

Continue....

Program Implementing Deadlock

Continue.

```
try { Thread.sleep(10); }

   catch (InterruptedException e) {}

System.out.println("Thread 2: Waiting for lock 1...");



    synchronized (Lock1) {

System.out.println("Thread 2: Holding lock 1 & 2...");

          }

        }

      }
```

Deadlock Solution

- Avoid Unnecessary Locks
- Avoid Nested Locks
- Using Thread.join() Method
- Use Lock Ordering
- Lock Time-out
- Just changing the order of the locks prevent the program in going into a deadlock situation

Stopping a Thread

- A thread is automatically destroyed when the run() method has completed.
- But it might be required to stop a thread before it has completed its life_cycle.
- Modern ways to suspend/stop a thread are by using a boolean flag and Thread.interrupt() method.

Using a boolean flag

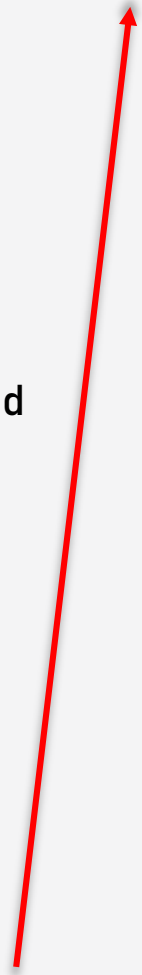
```
class MyThread implements Runnable {
    // to stop the thread
  private boolean exit;
    private String name;
  Thread t;
    MyThread(String threadname)
  {       name = threadname;
     t = new Thread(this, name);
  System.out.println("New thread: " + t);
     exit = false;
     t.start(); // Starting the thread
  }
```

```
// execution of thread starts from run() method
    public void run()
    {
        int i = 0;
        while (!exit) {
            System.out.println(name + ": " + i);
            i++;
            try {
                Thread.sleep(100);
            }
            catch (InterruptedException e) {
                System.out.println("Caught:" + e);
            }
        }
        System.out.println(name + " Stopped.");
    }
     // for stopping the thread
    public void stop()
    {
        exit = true;
    }
}
```

Continue....

Using a boolean flag

```
// Main class
public class Main {
    public static void main(String args[])
    {
        // creating two objects t1 & t2 of MyThread
        MyThread t1 = new MyThread("First thread");
        MyThread t2 = new MyThread("Second thread");
        try {
            Thread.sleep(500);
t1.stop(); // stopping thread t1
t2.stop(); // stopping thread t2
Thread.sleep(500);
            }
catch (InterruptedException e) {
System.out.println("Caught:" + e);
                }
 System.out.println("Exiting the main Thread");
                }
                }
```



Continue....

Using Thread.interrupt() method

```
class MyThread implements
Runnable {
    Thread t;
   MyThread()
   {
       t = new Thread(this);
       System.out.println("New
thread: " + t);
       t.start(); // Starting the thread
   }
// execution of thread starts
      from run() method
   public void run()
   {
while (!Thread.interrupted()) {
```

Continue....

Using Thread.interrupt() method

Continue....

```
System.out.println("Thread is running");
        }
        System.out.println("Thread has
stopped.");
    } }
// Main class
public class Main {
    public static void main(String args[])
    {
        // creating objects t1 of MyThread
        MyThread t1 = new MyThread();
          try {
            Thread.sleep(1);
```

Continue....

Using Thread.interrupt() method

Continue....

```
// t1 is an object of MyThread
            // which has an object t
            // which is of type Thread
            t1.t.interrupt();
           Thread.sleep(5);
        }
        catch (InterruptedException e) {
            System.out.println("Caught:" +
e);
        }
        System.out.println("Exiting the
main Thread");
    } }
```


That’s all for now…