What is Deadlock and how to avoid it ?

When two or more threads are waiting for each other to release lock and get stuck for infinite time, situation is called deadlock .

<http://tutorials.jenkov.com/java-concurrency/deadlock.html>

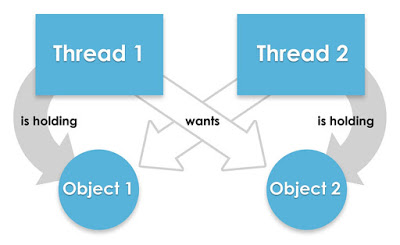
Deadlock is a programming situation where two or more threads are blocked forever

For instance, if thread 1 locks A, and tries to lock B, and thread 2 has already locked B, and tries to lock A, a deadlock arises. Thread 1 can never get B, and thread 2 can never get A. In addition, neither of them will ever know. They will remain blocked on each their object, A and B, forever. This situation is a deadlock.

The situation is illustrated below:

Thread 1 locks A, waits for B

Thread 2 locks B, waits for A



Write a program which will result in deadlock

<http://javarevisited.blogspot.com/2010/10/what-is-deadlock-in-java-how-to-fix-it.html>

/\*\*

\* Java program to create a deadlock by imposing circular wait.

\*

\* @author WINDOWS 8

\*

\*/

public class DeadLockDemo {

/\*

\* This method request two locks, first String and then Integer

\*/

public void method1() {

synchronized (String.class) {

System.out.println("Aquired lock on String.class object");

synchronized (Integer.class) {

System.out.println("Aquired lock on Integer.class object");

}

}

}

/\*

\* This method also requests same two lock but in exactly

\* Opposite order i.e. first Integer and then String.

\* This creates potential deadlock, if one thread holds String lock

\* and other holds Integer lock and they wait for each other, forever.

\*/

public void method2() {

synchronized (Integer.class) {

System.out.println("Aquired lock on Integer.class object");

synchronized (String.class) {

System.out.println("Aquired lock on String.class object");

}

}

}

}

If method1() and method2() both will be called by two or many threads , there is a good chance of deadlock because if thread 1 acquires lock on Sting object while executing method1() and thread 2 acquires lock on Integer object while executing method2() both will be waiting for each other to release lock on Integer and String to proceed further which will never happen.

Write code prevent deadlock

public class DeadLockFixed {

/\*\*

\* Both method are now requesting lock in same order, first Integer and then String.

\* You could have also done reverse e.g. first String and then Integer,

\* both will solve the problem, as long as both method are requesting lock

\* in consistent order.

\*/

public void method1() {

synchronized (Integer.class) {

System.out.println("Aquired lock on Integer.class object");

synchronized (String.class) {

System.out.println("Aquired lock on String.class object");

}

}

}

public void method2() {

synchronized (Integer.class) {

System.out.println("Aquired lock on Integer.class object");

synchronized (String.class) {

System.out.println("Aquired lock on String.class object");

}

}

}

}

Now there would not be any deadlock because both methods are accessing lock on Integer and String class literal in same order. So, if thread A acquires lock on Integer object , thread B will not proceed until thread A releases Integer lock, same way thread A will not be blocked even if thread B holds String lock because now thread B will not expect thread A to release Integer lock to proceed further.