**How do you detect deadlock in Java?**  
Answer is simple, when two or more threads are waiting for each other to release lock and get stuck for infinite time, situation is called deadlock. It will only happen in case of multitasking.  
  
**How do you detect deadlock in Java ?**

Though this could have many answers, my version is first I would look the code if I see nested synchronized block or calling one synchronized method from other or trying to get lock on different object then there is good chance of deadlock if developer is not very careful.  
  
Other way is to find it when you actually get locked while running the application , try to take thread dump , in Linux you can do this by command **"kill -3"** , this will print status of all the thread in application log file and you can see which thread is locked on which object.  
  
Other way is to use **jconsole**, it will show you exactly which threads are get locked and on which object.

**Write a Java program which will result in deadlock?**

Once you answer this , they may ask you to **write code which will result in deadlock ?**  
here is one of my version

/\*\*

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

\*/

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");

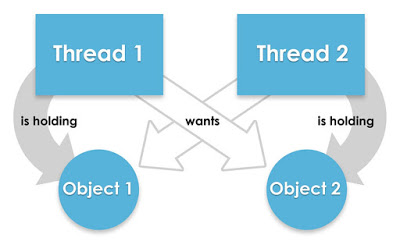
}

}

}

}

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.  
  
This diagram exactly demonstrate our program, where one thread holds lock on one object and waiting for other object lock which is held by other thread.

[](http://2.bp.blogspot.com/-63RZ-BTlAFs/VfGeHMnGdFI/AAAAAAAADuw/gwqtrVliMsM/s1600/Deadlock+of+Threads.jpg)

**How to avoid deadlock in Java?**

Now interviewer comes to final part, one of the most important in my view; *How do you fix deadlock?*or **How to avoid deadlock in Java?**  
  
If you have looked above code carefully then you may have figured out that real reason for deadlock is not multiple threads but ***the way they are requesting lock*** , if you provide an ordered access then problem will be resolved , here is my fixed version, which avoids deadlock by avoiding circular wait with no preemption.

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.  
  
Read more: <http://javarevisited.blogspot.com/2010/10/what-is-deadlock-in-java-how-to-fix-it.html#ixzz3uq2Les5W>