```
synchronized block
===============
eg:1
eg#1.
class Display{
      public void wish(String name){
            ;;;;;;;;;;;;; //l-lakh lines of code

      synchronized(this){
            for (int i=1;i<=10;i++ )
            {
                  System.out.print("Good morning:");
                  try{
                        Thread.sleep(2000);
                  }
                  catch (InterruptedException e){}
                  System.out.println(name);
            }
      }
          ;;;;;;;;;;;;;//1-lakh lines of code
      }
}
class MyThread extends Thread{
      Display d;
      String name;

      MyThread(Display d,String name){
            this.d=d;
            this.name=name;
      }

      public void run(){
            d.wish(name);
      }
}

class Test {
      public static void main(String[] args) {
            Display d=new Display();
            MyThread t1=new MyThread(d,"dhoni");
            MyThread t2=new MyThread(d,"yuvi");
            t1.start();
            t2.start();
      }
}


eg#2.
class Display{
      public void wish(String name){
            ;;;;;;;;;;;;; //l-lakh lines of code

      synchronized(this){
            for (int i=1;i<=10;i++ )
            {
                  System.out.print("Good morning:");
                  try{
                        Thread.sleep(2000);
                  }
                  catch (InterruptedException e){}
                  System.out.println(name);
            }
      }
          ;;;;;;;;;;;;;//1-lakh lines of code
      }
}
class MyThread extends Thread{
      Display d;
      String name;

      MyThread(Display d,String name){
            this.d=d;
            this.name=name;
      }

      public void run(){
            d.wish(name);
      }
}
public class Test {
      public static void main(String[] args) {
            Display d1=new Display();
            Display d2=new Display();
            MyThread t1=new MyThread(d1,"dhoni");
            MyThread t2=new MyThread(d2,"yuvi");
            t1.start();
            t2.start();
      }
}
```

Output::Irregular output becoz two object and two threads acting on two different objects

eg#3.

```
class Display{
      public void wish(String name){
            ;;;;;;;;;;;;; //l-lakh lines of code

      synchronized(Display.class){
            for (int i=1;i<=10;i++ )
            {
                  System.out.print("Good morning:");
                  try{
                        Thread.sleep(2000);
                  }
                  catch (InterruptedException e){}
                  System.out.println(name);
            }
      }
          ;;;;;;;;;;;;;//1-lakh lines of code
      }
}
class MyThread extends Thread{
      Display d;
      String name;

      MyThread(Display d,String name){
            this.d=d;
            this.name=name;
      }

      public void run(){
            d.wish(name);
      }
}
public class Test {
      public static void main(String[] args) {
            Display d1=new Display();
            Display d2=new Display();
            MyThread t1=new MyThread(d1,"dhoni");
            MyThread t2=new MyThread(d2,"yuvi");
            t1.start();
            t2.start();
      }
}
```

Note:: 2 object, 2 thread, but the thread which gets a chance applied class level lock so output is regular.

Note:: lock concept applicable only for objects and class types, but not for primitive types. if we try to do it would
result in compile time error saying "unexpected type".

```
eg:: int x=10;
            synchronized(x){//CE: unexpected type found:int required:reference
       }
```

InterThreadCommunication(remember postbox example)
==
Two threads can communicate each other with the help of
 a. notify()
 b. notifyAll()
 c. wait()

notify()=> Thread which is performing updation should call notify(),so the waiting thread will
get notification so it will continue with its execution with the updated items.
wait() => Thread which is expecting notification/updation should call wait(), immediately the
Thread will enter into waiting state.

If a thread wants to call wait(),notify()/notifyall() then compulsorily the thread should be
the owner of the object otherwise it would result in "IllegalMonitorStateException".

We say thread to be owner of that object if thread has lock of that object.

It means these methods are part of synchronized block or synchronized method, if we try to use
outside synchronized area then it would result in RunTimeException called "IllegalMonitorStateException".

if a thread calls wait() on any object, then first it immediately releases the lock on that object and it enters into waiting state.
if a thread calls notify() on any object,then he may or may not release the lock on that object immediately.

```
Except wait(),notify(),notifyAll() lock can't be relased by other methods.

Note::
yield(),sleep(),join() => can't release the lock.
wait(),notify(),notifyAll() => will release the lock,otherwise interthread
communication can't happen.

Once a Thread calls wait(), notify(), notifyAll() methods on any object then it
releases the lock of that particular object but
not all locks it has.

Method prototype of wait(),notify(),notifyAll()
1. public final void wait()throws InterruptedException
2. public final native void wait(long ms)throws InterruptedException
3. public final void wait(long ms,int ns)throws InterruptedException
4. public final native void notify()
5. public final void notifyAll()

Interview Question
================
Method like wait(),notify(),notifyAll() are present inside Object class, y not in
Thread class?
  Thread will call wait(),notify(),notifyAll() on Objects like
PostBox,Stack,Customer,Student,....
            => obj.wait(),obj.notify(),obj.notifyAll()
These methods should be available for every object in java,if the method has to
available for every object in java then those
methods should come from "Object" class.

Program
=======
eg#1.
class ThreadB extends Thread{
      int total =0;

      @Override
      public void run(){
            for (int i=0;i<=100 ; i++){
                  total+=i;
            }
      }
}
public class Test {
      public static void main(String[] args)throws InterruptedException {
            ThreadB b=new ThreadB();
            b.start();

            stmt-1;
            System.out.println(b.total);
      }
}

A. stmt-1
      if i replace with Thread.sleep(10000) then thread will enter into waiting
statement
        but within 1ns only the updation value is ready.
        with in 10 sec if the updation is not ready, then we should not use
Thread.sleep(10000)
```

```
B. stmt-2
       if i replace with b.join(), then main thread will enter into waiting
state,then child will
       execute for loop,till then main thread has to wait.
       main thread is waiting for updation result.
            for (int i=0;i<=100 ; i++){
                  total+=i;
          }
            //1cr lines of code is available
        main thread has to wait till 1 cr lines of code,y main thread should wait
for the
        complete code to finish.


eg#2.
class ThreadB extends Thread{
      int total =0;

  @Override
  public void run(){

      synchronized(this){
            System.out.println("Child thread started calculation");
            for (int i=0;i<=100 ; i++){
                  total+=i;
            }
            System.out.println("Child thread trying to give notification");
            this.notify();
      }
  }
}
public class Test {
      public static void main(String[] args)throws InterruptedException {
            ThreadB b=new ThreadB();
            b.start();

            Thread.sleep(10000);//10sec

      synchronized(b){
            System.out.println("Main thread is calling wait on B object");
            b.wait();
            System.out.println("Main thread got notification");
            System.out.println(b.total);
      }

   }
}
Output
======
Child thread started calculation
Child thread trying to give notification
Main thread is calling wait on B object
  becoz of Thread.sleep(10000) main thread will never get notification.

eg#3.

class ThreadB extends Thread{
      int total =0;
  @Override
  public void run(){

      synchronized(this){
            System.out.println("Child thread started calculation");//step-2
            for (int i=0;i<=100 ; i++){
                  total+=i;
            }
            System.out.println("Child thread trying to give notification");//step-3
            this.notify();
      }
  }
}
public class Test {
      public static void main(String[] args)throws InterruptedException {
            ThreadB b=new ThreadB();
            b.start();

      synchronized(b){
            System.out.println("Main thread is calling wait on B object");//step-1
            b.wait(10000);//10sec
            System.out.println("Main thread got notification");//step-4
            System.out.println(b.total);
      }

  }
}
Output
======
Child thread started calculation
Child thread trying to give notification
Main thread is calling wait on B object for 10sec
Main thread got notification
5050

ProducerConsumer Problem
========================
  Producer => produce the item and update in the Queue
  Consumer => consume the item from the Queue


class Producer extends Thread{
      Producer(){
            synchronized(q){
                  produce the item and update it to queue
                  q.notify();
            }
      }
}
class Consumer extends Thread{
      Consumer(){
            synchronized(q){
                  if(q is empty){
                        q.wait();
                  }else{
                        consume the item from the queue
                  }
            }
      }
```

```
}
```

Difference b/w notify and notifyAll()

```
 notify()    => To give notification only for one waiting thread
 notifyAll() => To give notification for many waiting thread
```

=> We can use notify() method to give notification for only one Thread. If multiple
   Threads are waiting then only one Thread will get the chance and remaining
Threads has to wait   for further notification.
   But which Thread will be notify(inform) we can't expect exactly it depends on
JVM.

```
eg::                        waiting state
                                 |
                                 |    obj1.wait(); 60 threads are waiting
       obj1.notify()  |
                                 |
                     Running state
```

Among 60 threads which thread will get a chance we don't have control over that it
is decided
by JVM(threadscheduler).

=> We can use notifyAll() method to give the notification for all waiting Threads
of particular object.
      All waiting Threads will be notified and will be executed one by one, because
they required lock.

```
 eg::                     waiting state
                                 |
                                 |    obj1.wait(); 60 threads are waiting
   obj1.notifyAll() |    obj2.wait(); 40 threads are waiting
                                 |
                     Running state
```

Note: On which object we are calling wait(), notify() and notifyAll() methods that
           corresponding object lock we have to get but not other object locks.

```
eg:: Stack s1=new Stack();
     Stack s2=new Stack();

synchronized(s1){
      s2.wait();//RE: IllegalMonitorStateException
}

synchronized(s2){
      s2.wait();(valid)
}
```

Question based on lock
======================
1. If a thread calls wait() immediately it will enter into waiting state without
releasing any lock.(false)
2. If a thread calls wait() it releases the lock of that object but may not
immediately (false)
3. If a thread calls wait() on any object,it releases all locks acquired by that
thread and enters into waiting state(false)
4. If a thread calls wait() on any object,it immediately releases the lock of that
particular object and entered into
     waiting state(true).
5. If a thread calls notify() on any object,it immediately releases the lock of

that particular object(invalid)
6. If a thread calls notify() on any object,it releases the lock of that object but may not immediately(true)