Methods of Object class to achieve ITC :
----

public final void wait() throws InterruptedException :-

It is a predefined non static method of Object class. We can use this method from synchronized area only otherwise we will get java.lang.IllegalMonitorStateException.

It will put a thread into temporarly waiting state and it will release the Object lock, It will remain in the waiting state till another thread sends a notification message on the same object, After getting the notification the waiting thread will move from WAITING to BLOCKED state and now it is waiting for Object lock to re-enter inside synchronized area. Once the lock is available, It will move to RUNNABLE State to get processor time.

It throws a checked Exception i.e InterruptedException.

public native final void notify() :-

It will wake up the single thread that is waiting on the same object.It will not release the lock , once synchronized area is completed then only lock will be released.

Once a waiting thread will get the notification from the another thraed using notify()/notifyAll() method on the same object then the waiting thread will move to Blocked state and once it will get the lock then it will come to Runnable state, now It depends upon the Thread scheduler to schedule this thread for execution.

public native final void notifyAll() :-

It will wake up all the threads which are waiting on the same object.It will not release the lock , once synchronized area is completed then only lock will be released.

ITC Flow :

Step	Description	Thread State
A thread calls `wait()` inside a synchronized block	Releases lock and goes to sleep	`WAITING`
Another thread calls `notify()`	Wakes up **one** waiting thread	(Waiting thread moves internally to **BLOCKED**)
Notified thread tries to reacquire the lock	Still **BLOCKED** (waiting for the lock)	
Lock becomes available	Thread moves to **RUNNABLE**	
Scheduler picks the thread	Thread becomes **RUNNING**	

//Programs :
----
1) WAP to show wait(), notify() and notifyAll() must be used from synchronized area.

```java
package com.ravi.itc;

public class ITCDemo1
{
    public static void main(String[] args)
    {
        System.out.println("Main Thread started!!!");
        try
        {
            Object obj = new Object();
            obj.wait();
        }
        catch(InterruptedException e)
        {
            System.err.println("Thread is Interrupted");
        }
    }
}
```

2) /*Program that describes, If we don't have proper communication between two
    threads then output is un-predictable */

```java
package com.ravi.itc;

//Program that describes, If we don't have proper communication between two
//threads then output is un-predictable

class Test implements Runnable
{
    private int data = 0;

    @Override
    public void run()
    {
        try
        {
            Thread.sleep(5);
        }
        catch(InterruptedException e) {}

        for(int i=1; i<=10; i++)
        {
            data = data + i;     //1   3    6     10
        }
    }

    public int getData()
    {
        return this.data;
    }
}

public class ITCDemo2
{
    public static void main(String[] args) throws InterruptedException
    {
        System.out.println("Main Thread stated!!");

        Test t1 = new Test();

        Thread thread = new Thread(t1);
        thread.start();

        Thread.sleep(10);

        System.out.println(t1.getData());

        System.out.println("Main Thread ended!!");

    }

}
```

3) WAP to show by using ITC we can get the proper output

```java
package com.ravi.itc;

class Demo extends Thread
{
    private int val = 0;

    @Override
    public void run()
    {

        synchronized(this)
        {
            System.out.println("Child Thread got the lock...");

            for(int i=1; i<=100; i++)
            {
                val = val + i;
            }
            System.out.println("Child Thread is sending notification...");
            notify();

        }
    }

    public int getVal()
    {
        return this.val;
    }
}
public class ITCDemo3
{
    public static void main(String[] args) throws InterruptedException
    {
        Demo d1 = new Demo();
        d1.start();

        synchronized (d1)
        {
            System.out.println("Main thread is going to wait...");
            d1.wait(); //Main thread has released the lock
            System.out.println("Main thread got notification");
            System.out.println(d1.getVal());
        }

    }
}
```

4) WAP to withdraw the amount after successful deposit if amount is less.

```java
package com.ravi.itc;

class Customer
{
    private double balance;

    public Customer(double balance)
    {
        super();
        this.balance = balance;
    }

    public synchronized void withdraw(double amount)
    {
        System.out.println("Going for withdraw!!!");
        if(amount > this.balance)
        {
            System.out.println("Low Balance, Waiting for deposit");
            try
            {
                wait();
            }
            catch(InterruptedException e)
            {
                System.err.println("Withdraw thread interrupted");
            }
        }
        this.balance = this.balance - amount;
        System.out.println("Amount after withdraw is :"+this.balance);

    }


    public synchronized void deposit(double amount)
    {
        System.out.println("Going for deposit!!");
        this.balance = this.balance + amount;
        System.out.println("Balance after deposit is :"+this.balance);
        notify();
    }

}


public class ITCDemo4
{
    public static void main(String[] args)
    {
        Customer customer = new Customer(10000); //lock is created

        Thread son = new Thread()
        {
            @Override
            public void run()
            {
                customer.withdraw(15000);
            }

        };

        son.start();

        Thread father = new Thread()
        {
            @Override
            public void run()
            {
                customer.deposit(10000);
            }

        };

        father.start();

    }

}
```

```java
package com.ravi.itc;

class TicketSystem
{
    private int availableTicket = 5;  //availableTicket = 5

    public synchronized void bookTicket(int numberOfTickets) //numberOfTickets = 5
    {
        while(numberOfTickets > availableTicket)
        {
            System.err.println("Tickets are not available, Waiting for cancellation");

            try
            {
                wait();
            }
            catch(InterruptedException e)
            {
                System.err.println("booking thread has interrupted!!!");
            }

        }

        this.availableTicket = this.availableTicket - numberOfTickets;

        System.out.println("Booked "+numberOfTickets+" Ticket(s), Available tickets are
:"+this.availableTicket);
        notify();
    }


    public synchronized void cancelTicket(int numberOfTickets) //numerOfTickets = 2
    {
        this.availableTicket = this.availableTicket + numberOfTickets;
        System.out.println("Cancel "+numberOfTickets+" Ticket(s), Available ticket is
"+this.availableTicket);
        notify();
    }
}

public class ITCDemo5
{
    public static void main(String[] args)
    {
        TicketSystem ticketSystem = new TicketSystem();


        Thread bookingThread = new Thread()
        {
            @Override
            public void run()
            {
                int tickets [] = {4, 2, 5};

                for(int ticket : tickets)
                {
                    ticketSystem.bookTicket(ticket);  //3rd Iteration [Ticket = 5]
                    try
                    {
                        Thread.sleep(1000);
                    }
                    catch(InterruptedException e)
                    {
                        System.err.println("Booking Thread interrupted!!!");
                    }
                }

            }
        };

        bookingThread.start();


        Thread cancelThread = new Thread()
        {
            @Override
            public void run()
            {
                int []tickets = {1,3,2};  //3rd Iteration [Ticket = 2]

                for(int ticket : tickets)
                {
                    ticketSystem.cancelTicket(ticket);
                    try
                    {
                        Thread.sleep(1000);
                    }
                    catch(InterruptedException e)
                    {
                        System.err.println("Cancel Thread interrupted!!!");
                    }
                }

            }
        };

        cancelThread.start();

    }

}
```