class MultithreadingDemo extends Thread

{

    public void run()

    {

        try

        {

            // Displaying the thread that is running

            System.out.println ("Thread " +

                  Thread.currentThread().getId() +

                  " is running");

        }

        catch (Exception e)

        {

            // Throwing an exception

            System.out.println ("Exception is caught");

        }

    }

}

// Main Class

public class Multithread

{

    public static void main(String[] args)

    {

        int n = 8; // Number of threads

        for (int i=0; i<n; i++)

        {

            MultithreadingDemo object = new MultithreadingDemo();

            object.start();

        }

    }

}

Another method

class MultithreadingDemo implements Runnable

{

    public void run()

    {

        try

        {

            // Displaying the thread that is running

            System.out.println ("Thread " +

                                Thread.currentThread().getId() +

                                " is running");

        }

        catch (Exception e)

        {

            // Throwing an exception

            System.out.println ("Exception is caught");

        }

    }

}

// Main Class

class Multithread

{

    public static void main(String[] args)

    {

        int n = 8; // Number of threads

        for (int i=0; i<n; i++)

        {

            Thread object = new Thread(new MultithreadingDemo());

            object.start();

        }

    }

}

**Locking Example-Object level🡺 Task 1.4**

package com.test;

public class Foo implements Runnable {

@Override

public void run() {

Lock();

}

public void Lock() {

System.out.println(Thread.currentThread().getName());

synchronized(this) {

System.out.println("in block " + Thread.currentThread().getName());

System.out.println("in block " + Thread.currentThread().getName() + " end");

}

}

public static void main(String[] args) {

Foo b1 = new Foo();

Thread t1 = new Thread(b1);

Thread t2 = new Thread(b1);

Foo b2 = new Foo();

Thread t3 = new Thread(b2);

t1.setName("t1");

t2.setName("t2");

t3.setName("t3");

t1.start();

t2.start();

t3.start();

}

}

Another locking example (class level)

package com.test;

public class Foo implements Runnable {

@Override

public void run() {

Lock();

}

public void Lock() {

System.out.println(Thread.currentThread().getName());

synchronized(Foo.class) {

System.out.println("in block " + Thread.currentThread().getName());

System.out.println("in block " + Thread.currentThread().getName() + " end");

}

}

public static void main(String[] args) {

Foo b1 = new Foo();

Thread t1 = new Thread(b1);

Thread t2 = new Thread(b1);

Foo b2 = new Foo();

Thread t3 = new Thread(b2);

t1.setName("t1");

t2.setName("t2");

t3.setName("t3");

t1.start();

t2.start();

t3.start();

}

}

package com.journaldev.[concurrency](https://www.journaldev.com/1162/java-multithreading-concurrency-interview-questions-answers);

public class Message {

private String msg;

public Message(String str){

this.msg=str;

}

public String getMsg() {

return msg;

}

public void setMsg(String str) {

this.msg=str;

}

}

**Waiter**

A class that will wait for other threads to invoke notify methods to complete it’s processing. Notice that Waiter thread is owning monitor on Message object using synchronized block.

package com.journaldev.concurrency;

public class Waiter implements Runnable{

private Message msg;

public Waiter(Message m){

this.msg=m;

}

@Override

public void run() {

String name = Thread.currentThread().getName();

synchronized (msg) {

try{

System.out.println(name+" waiting to get notified at time:"+System.currentTimeMillis());

msg.wait();

}catch(InterruptedException e){

e.printStackTrace();

}

System.out.println(name+" waiter thread got notified at time:"+System.currentTimeMillis());

//process the message now

System.out.println(name+" processed: "+msg.getMsg());

}

}

}

### Notifier

A class that will process on Message object and then invoke notify method to wake up threads waiting for Message object. Notice that synchronized block is used to own the monitor of Message object.

package com.journaldev.concurrency;

public class Notifier implements Runnable {

private Message msg;

public Notifier(Message msg) {

this.msg = msg;

}

@Override

public void run() {

String name = Thread.currentThread().getName();

System.out.println(name+" started");

try {

Thread.sleep(1000);

synchronized (msg) {

msg.setMsg(name+" Notifier work done");

msg.notify();

// msg.notifyAll();

}

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

### WaitNotifyTest

Test class that will create multiple threads of Waiter and Notifier and start them.

package com.journaldev.concurrency;

public class WaitNotifyTest {

public static void main(String[] args) {

Message msg = new Message("process it");

Waiter waiter = new Waiter(msg);

new Thread(waiter,"waiter").start();

Waiter waiter1 = new Waiter(msg);

new Thread(waiter1, "waiter1").start();

Notifier notifier = new Notifier(msg);

new Thread(notifier, "notifier").start();

System.out.println("All the threads are started");

}

}