

# Synchronization in Java

Difficulty Level : Medium Last Updated : 09 Dec, 2021

Multi-threaded programs may often come to a situation where multiple threads try to access the same resources and finally produce erroneous and unforeseen results.

So it needs to be made sure by some synchronization method that only one thread can access the resource at a given point in time. Java provides a way of creating threads and synchronizing their tasks using synchronized blocks. Synchronized blocks in Java are marked with the synchronized keyword. A synchronized block in Java is synchronized on some object. All synchronized blocks synchronize on the same object can only have one thread executing inside them at a time. All other threads attempting to enter the synchronized block are blocked until the thread inside the synchronized block exits the block.

Following is the general form of a synchronized block:

```
// Only one thread can execute at a time.  
// sync_object is a reference to an object  
// whose lock associates with the monitor.  
// The code is said to be synchronized on  
// the monitor object  
synchronized(sync_object)  
{  
    // Access shared variables and other  
    // shared resources  
}
```

This synchronization is implemented in Java with a concept called monitors. Only one thread can own a monitor at a given time. When a thread acquires a lock, it is said to have entered the monitor. All other threads attempting to enter the locked monitor will be suspended until the first thread exits the monitor.

Following is an example of multi-threading with synchronized.

---

```
// A Java program to demonstrate working of
// synchronized.

import java.io.*;
import java.util.*;

// A Class used to send a message
class Sender
{
    public void send(String msg)
    {
        System.out.println("Sending\t" + msg );
        try
        {
            Thread.sleep(1000);
        }
        catch (Exception e)
        {
            System.out.println("Thread interrupted.");
        }
        System.out.println("\n" + msg + "Sent");
    }
}

// Class for send a message using Threads
class ThreadedSend extends Thread
{
    private String msg;
    Sender sender;

    // Receives a message object and a string
    // message to be sent
    ThreadedSend(String m, Sender obj)
    {
        msg = m;
        sender = obj;
    }

    public void run()
    {
        // Only one thread can send a message
        // at a time.
        synchronized(sender)
        {
            // synchronizing the snd object
            sender.send(msg);
        }
    }
}

// Driver class
class SyncDemo
{
    public static void main(String args[])
    {
        Sender snd = new Sender();
    }
}
```

```
ThreadedSend S1 =
    new ThreadedSend( " Hi " , snd );
ThreadedSend S2 =
    new ThreadedSend( " Bye " , snd );

// Start two threads of ThreadedSend type
S1.start();
S2.start();

// wait for threads to end
try
{
    S1.join();
    S2.join();
}
catch(Exception e)
{
    System.out.println("Interrupted");
}
}
```

## Output

```
Sending      Hi

Hi Sent
Sending      Bye

Bye Sent
```

The output is the same every time we run the program.

In the above example, we choose to synchronize the Sender object inside the run() method of the ThreadedSend class. Alternately, we could define the **whole send() block as synchronized**, producing the same result. Then we don't have to synchronize the Message object inside the run() method in ThreadedSend class.

```
// An alternate implementation to demonstrate
// that we can use synchronized with method also.
```

```
class Sender {
    public synchronized void send(String msg)
    {
```

```
        System.out.println("Sending\t" + msg);
        try {
            Thread.sleep(1000);
        }
        catch (Exception e) {
            System.out.println("Thread interrupted.");
        }
        System.out.println("\n" + msg + "Sent");
    }
}
```

We do not always have to synchronize a whole method. Sometimes it is preferable to **synchronize only part of a method**. Java synchronized blocks inside methods make this possible.

```
// One more alternate implementation to demonstrate
// that synchronized can be used with only a part of
// method

class Sender
{
    public void send(String msg)
    {
        synchronized(this)
        {
            System.out.println("Sending\t" + msg );
            try
            {
                Thread.sleep(1000);
            }
            catch (Exception e)
            {
                System.out.println("Thread interrupted.");
            }
            System.out.println("\n" + msg + "Sent");
        }
    }
}
```

This article is contributed by **Souradeep Barua**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above