Importance of Thread Synchronization in Java

Difficulty Level : Easy • Last Updated : 07 Jan, 2022

Our systems are working in a multithreading environment that becomes an important part for OS to provide better utilization of resources. The process of running two or more parts of the program simultaneously is known as Multithreading. A program is a set of instructions in which multiple processes are running and within a process, multiple threads are working. Threads are nothing but lightweight processes. For example, in the computer we are playing video games at the same time we are working with MS word and listen to music. So, these are the processes we are working on concurrently. In this, every application has multiple sub-processes i.e. threads. In the previous example, we listen to music in which we are having a music player as an application that contains multiple sub-processes which are running like managing playlist, accessing the internet, etc. So, threads are the task to be performed and multithreading is multiple tasks/processes getting executed at the same time in the same program.

This is the basic introduction of multithreading which will further help to understand the importance of thread synchronization.

Thread Priorities

In java, every thread is assigned with a priority that determines how the threads should be treated with respect to each other. Thread's priority is used to decide when to switch from one running thread to the next. A higher priority thread can preempt a lower priority thread and may take more CPU time. In a simple way, a thread with gher priority gets the resource first as compared to the thread with lower priority. But, in case, when two threads with the same priority want the same resource then the situation becomes more complicated. So, in a multithreading environment, if printer. At the same time, another computer wants the printer to print its document. So, two computers are demanding the same resource i.e. printer. So if both the processes running together then the printer will print the document of one as well as of another computer. This will produce invalid output. Now, the same thing happens in the case of threads if two threads with the same priority or want the same resource leads to inconsistent output.

In java, when two or more threads try to access the same resource simultaneously it causes the java runtime to execute one or more threads slowly, or even suspend their execution. In order to overcome this problem, we have thread synchronization.

Synchronization means coordination between multiple processes/threads.

Types of synchronization:

There are two types of synchronization that are as follows:

1. Process synchronization
2. Thread synchronization

Here we will be mainly focusing on thread synchronization.

Thread synchronization basically refers to The concept of one thread execute at a time and the rest of the threads are in waiting state. This process is known as thread synchronization. It prevents the thread interference and inconsistency problem.

Synchronization is build using locks or monitor. In Java, a monitor is an object that is used as a mutually exclusive lock. Only a single thread at a time has the right to own a monitor. When a thread gets a lock then all other threads will get suspended which are trying to acquire the locked monitor. So, other threads are said to be waiting for the monitor, until the first thread exits the monitor. In a simple way, when a thread request a resource then that resource gets locked so that no other thread can work or do any modification until the resource gets released.

read Synchronization are of two types:

1. **Mutual Exclusive**
2. **Inter-Thread Communication**

- Synchronized Method
- Synchronized Block
- Static Synchronization

Synchronized Method

We can declare a method as synchronized using the *"synchronized"* keyword. This will make the code written inside the method thread-safe so that no other thread will execute while the resource is shared.

Implementation:

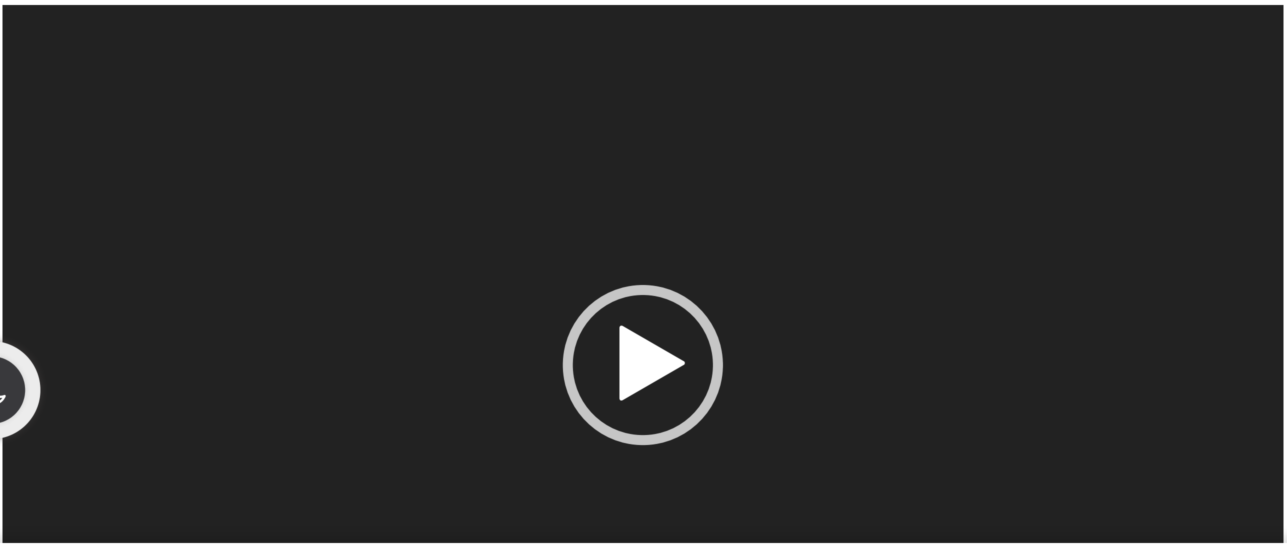
We will be proposing prints the two threads simultaneously showing the asynchronous behavior without thread synchronization.

Example 1:

```java
// Class 1
// Helper class
// Extending Thread class
public class PrintTest extends Thread {

    // Non synchronized Code

    // Method 1
    public void printThread(int n)
    {

        // This loop will print the  currently executed
        // thread
        for (int i = 1; i <= 10; i++) {
            System.out.println("Thread " + n
                               + " is working...");

            // Try block to check for exceptions
            try {

                // Pause the execution of current thread
                // for 0.600 seconds using sleep() method
                Thread.sleep(600);
            }
```

```
                System.out.println(ex.toString());
            }
        }

        // Display message for better readability
        System.out.println("--------------------------");

        try {

            // Pause the execution of current  thread
            // for 0.1000 millisecond or 1sec using sleep
            // method
            Thread.sleep(1000);
        }

        catch (Exception ex) {

            // Printing the exception
            System.out.println(ex.toString());
        }
    }
}

// Class 2
// Helper class extending Thread Class
public class Thread1 extends Thread {

    // Declaring variable of type Class1
    PrintTest test;

    // Constructor for class1
    Thread1(PrintTest p) { test = p; }

    // run() method of this class for
    // entry point for thread1
    public void run()
    {

        // Calling method  1 as in above class
        test.printThread(1);
    }
}

  Class 3
// Helper class extending Thread Class
public class Thread2 extends Thread {
    // run() method of this class for
    // entry point for thread2
    public void run() { test.printThread(2); }
}

// Class 4
// Main class
public class SynchroTest {

    // Main driver method
    public static void main(String[] args)
    {

        // Creating object of class 1 inside main() method
        PrintTest p = new PrintTest();

        // Passing the same object of class PrintTest to
        // both threads
        Thread1 t1 = new Thread1(p);
        Thread2 t2 = new Thread2(p);

        // Start executing the threads
        // using start() method
        t1.start();
        t2.start();

        // This will print both the threads  simultaneously
    }
}
```

Output:

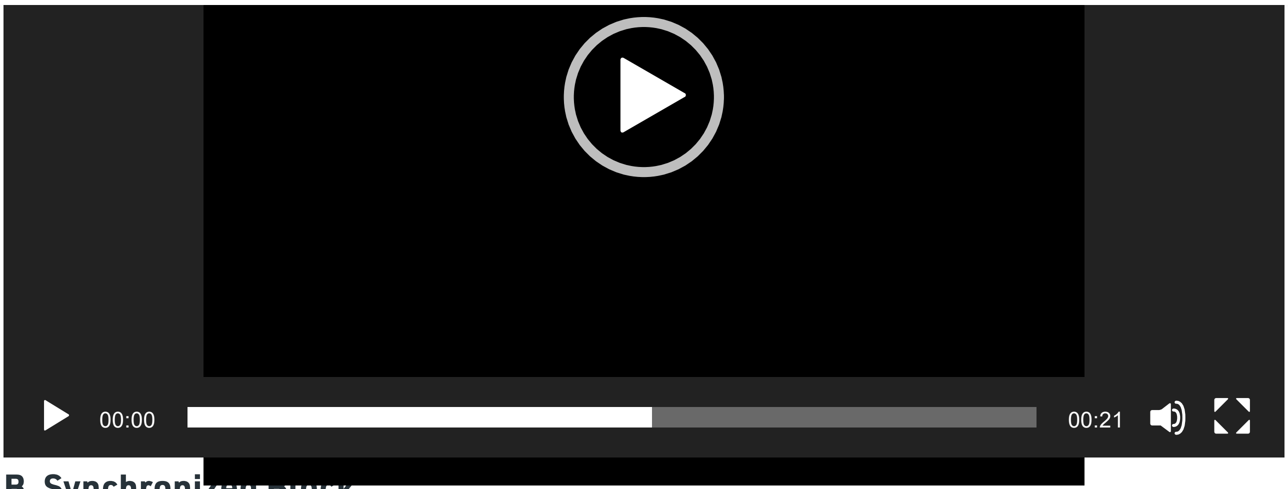


[redacted]

```java
// Java Program Illustrating Lock the Object for
// the shared resource giving consistent output

// Class 1
// Helper class extending Thread class
public class PrintTest extends Thread {

    // synchronized code
    // synchronized method will lock the object and
    // releases when thread is terminated or completed its
    // execution.
    synchronized public void printThread(int n)
    {
        for (int i = 1; i <= 10; i++) {
            System.out.println("Thread " + n
                               + " is working...");

            try {

                // pause the execution of current  thread
                // for 600 millisecond
                Thread.sleep(600);
            }
            catch (Exception ex) {
                // overrides toString() method  and prints
                // exception if occur
                System.out.println(ex.toString());
            }
        }
        System.out.println("--------------------------");
        try {

            // pause the execution of current  thread for
            // 1000 millisecond
            Thread.sleep(1000);
```

```
// creating thread1 extending Thread Class

public class Thread1 extends Thread {

    PrintTest test;
    Thread1(PrintTest p) { test = p; }

    public void run() // entry point for thread1
    {

        test.printThread(1);
    }
}
// creating thread2 extending Thread Class

public class Thread2 extends Thread {

    PrintTest test;
    Thread2(PrintTest p) { test = p; }
    public void run() // entry point for thread2
    {
        test.printThread(2);
    }
}

public class SynchroTest {

    public static void main(String[] args)
    {

        PrintTest p = new PrintTest();

        // passing the same object of class PrintTest to
        // both threads
        Thread1 t1 = new Thread1(p);
        Thread2 t2 = new Thread2(p);

        // start function will execute the threads
        t1.start();
        t2.start();
    }
}
```

Output:



B. Synchronized Block

If we declare a block as synchronized, only the code which is written inside that block is executed sequentially not the complete code. This is used when we want sequential access to some part of code or to synchronize some part of code.

Syntax:

```
synchronized (object reference)
{
   // Insert code here
}
```

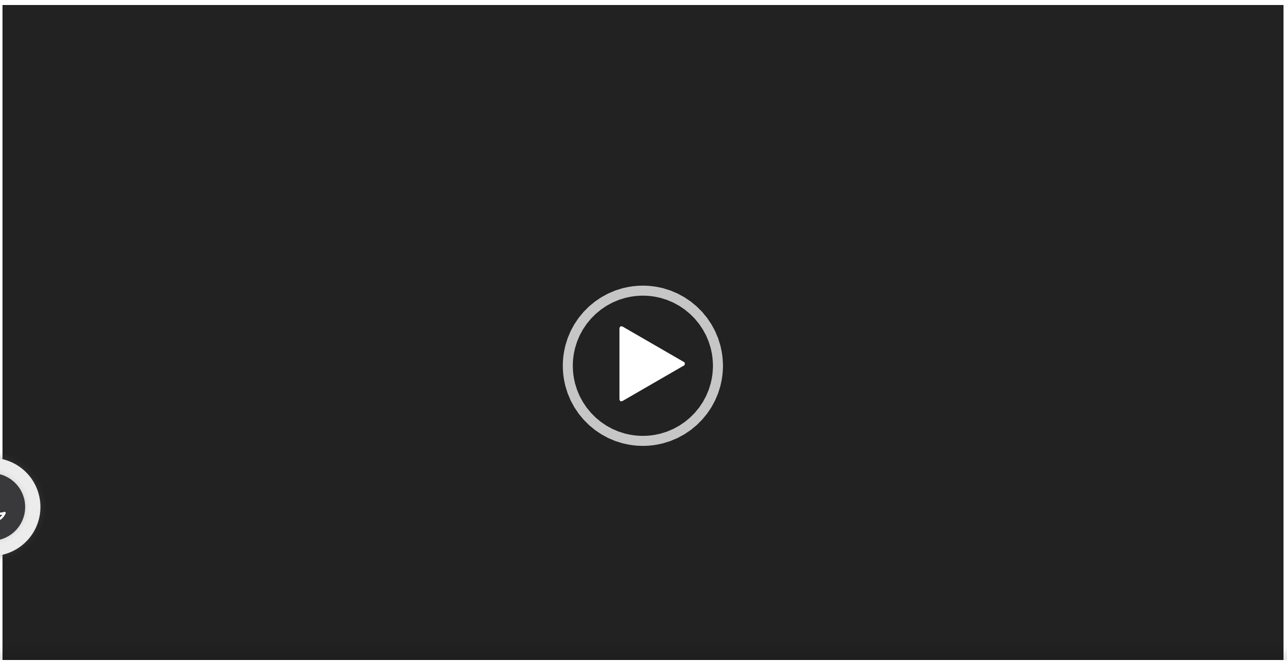
Example

```
// Java Program Illustrating  Synchronized Code
// Using synchronized block

// Class 1
// Helper class extending Thread class
class PrintTest extends Thread {

    // Method  1
    // To print the thread
    public void printThread(int n)
    {

        // Making synchronized block that makes the block
        // synchronized
```

```
                // Print message when these thread are
                // executing
                System.out.println("Thread " + n
                                   + " is working...");

                // Try block to check for exceptions
                try {

                    // Making thread to pause for 0.6
                    // seconds
                    Thread.sleep(600);
                }

                // Catch block to handle exceptions
                catch (Exception ex) {

                    // Print message when exception.s occur
                    System.out.println(ex.toString());
                }
            }
        }

        // Display message only
        System.out.println("--------------------------");

        try {

            // Making thread t osleep for 1 sec
            Thread.sleep(1000);
        }

        catch (Exception ex) {

            System.out.println(ex.toString());
        }
    }
}

// Class 2
// Helper class extending Thread class
class Thread1 extends Thread {

    PrintTest test;
    Thread1(PrintTest p) { test = p; }

    public void run() { test.printThread(1); }
}
```

```
    Thread2(PrintTest p) { test = p; }

    public void run() { test.printThread(2); }
}

// Class 4
// Main class
class SynchroTest {

    // Main driver method
    public static void main(String[] args)
    {

        // Creating instance for class 1 inside main()
        PrintTest p = new PrintTest();

        // Creating threads and
        // passing same object
        Thread1 t1 = new Thread1(p);
        Thread2 t2 = new Thread2(p);

        // Starting these thread using start() method
        t1.start();
        t2.start();
    }
}
```

Output:



th [redacted]

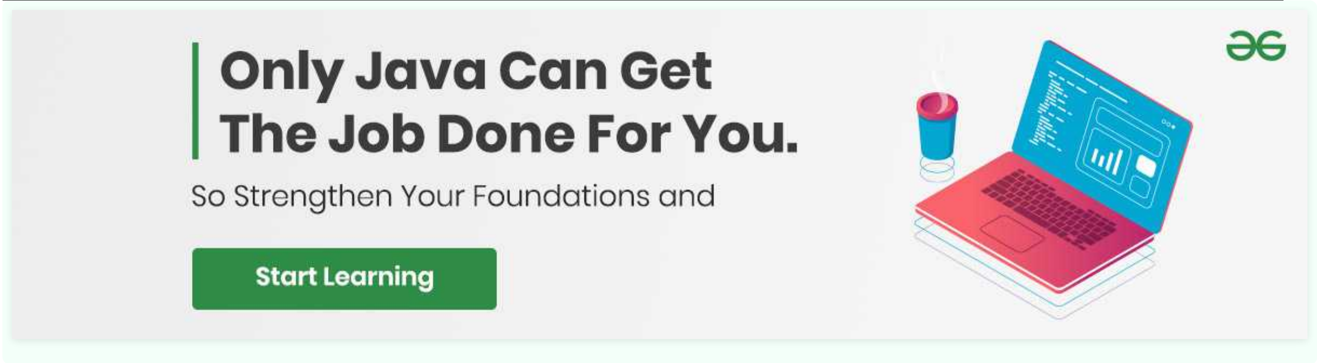
Ex

ava

```
// Java Program Illustrate  Synchronized
// Using static synchronization

// Class 1
// Helper class
class PrintTest extends Thread {

    // Static synchronization locks the class PrintTest
    synchronized public static void printThread(int n)
    {

        for (int i = 1; i <= 10; i++) {

            // Print message when threads are executing
            System.out.println("Thread " + n
                               + " is working...");

            // Try block to check for exceptions
            try {

                // making thread to sleep for 0.6 seconds
                Thread.sleep(600);
            }

            // Catch block to handle the exceptions
            catch (Exception ex) {

                // Print message when exception occurs
                System.out.println(ex.toString());
            }
        }

        // Display message for better readability
        System.out.println("--------------------------");

        try {
            Thread.sleep(1000);
        }
```

```
// Class 2
// Helper class extending Thread class
class Thread1 extends Thread {

    // run() method for thread
    public void run()
    {

        // Passing the class not the object
        PrintTest.printThread(1);
    }
}

// Class 3
// Helper class extending Thread class
class Thread2 extends Thread {

    public void run()
    {

        // Passing the class not the object
        PrintTest.printThread(2);
    }
}

// Class 4
// Main class
class SynchroTest {

    // Main driver method
    public static void main(String[] args)
    {

        // No shared object
        // Creating objects of class 2 and 3 that
        // are extending to Thr3ead class
        Thread1 t1 = new Thread1();
        Thread2 t2 = new Thread2();

        // Starting thread with help of start() method
        t1.start();
        t2.start();
    }
```

Start Your Coding Journey Now! Login Register



Like 9

Previous Next

RECOMMENDED ARTICLES Page : 1 2 3

Why Thread.stop(), Naming a thread and fetching

Start Your Coding Journey Now! Login Register

02 the uses of static synchronization in multithreading
21, Sep 21

in Java
19, Dec 17

03 Lock framework vs Thread synchronization in Java
27, Feb 19

07 Method and Block Synchronization in Java
20, Feb 17

04 Difference between Thread.start() and Thread.run() in Java
23, Jan 19

08 Java notify() Method in Threads Synchronization with Examples
13, Jun 21

Article Contributed By :

raismily01
@raismily01

Vote for difficulty

Current difficulty : Easy

Easy Normal Medium Hard Expert

Improved By : simmytarika5, surindertarika1234, nnr223442, anikakapoor, surinderdawra388, sweetyty, sagartomar9927, germanshephered48, arorakashish0911

ticle Tags : Java-Multithreading, Java

Practice Tags : Java