Method and Block Synchronization in Java

Difficulty Level : Easy Last Updated : 16 Oct, 2019

Threads communicate primarily by sharing access to fields and the objects reference fields refer to. This form of communication is extremely efficient, but makes two kinds of errors possible: thread interference and memory consistency errors. Some synchronization constructs are needed to prevent these errors. Following example shows a situation where we need synchronization.

Need of Synchronization

Consider the following Example:

```
// Java program to illustrate need
// of Synchronization
import java.io.*;

class Multithread
{
    private int i = 0;
    public void increment()
    {
        i++;
    }

    public int getValue()
    {
        return i;
    }
}

class GfG
{
    public static void main (String[] args)
    {
        Multithread t = new Multithread();
        t.increment();
        System.out.println(t.getValue());
    }
}
```

Output:

```
1
```

In above example three operations are performed:

1. Fetch the value of variable i.
2. Increment the fetched value.
3. And store the increased value of i to its location.

Here,

- 1st thread fetches the value of i. (Currently value i is 0) and increases it by one, so value of variable i becomes 1.
- Now 2nd thread accesses the value of i that would be 0 as 1st thread did not store it back to its location.
  And 2nd thread also increment it and store it back to its location. And 1st also store it.
- Finally value of variable i is 1. But it should be 2 by the effect of both threads. That's why we need to synchronize the access to shared variable i.

Java is multi-threaded language where multiple threads runs parallel to complete their execution. We need to synchronize the shared resources to ensure that at a time only one thread is able to access the shared resource.
If an Object is shared by multiple threads then there is need of synchronization in order to avoid the Object's state to be getting corrupted. Synchronization is needed when Object is mutable. If shared Object is immutable or all the threads which share the same Object are only reading the Object's state not modifying then you don't need to synchronize it.

Java programming language provide two synchronization idioms:

- Methods synchronization
- Statement(s) synchronization (Block synchronization)

Method Synchronization

Synchronized methods enables a simple strategy for preventing the thread interference and memory consistency errors. If a Object is visible to more than one threads, all reads or writes to that Object's fields are done through the **synchronized** method.

It is not possible for two invocations for synchronized methods to interleave. If one thread is executing the synchronized method, all others thread that invoke synchronized method on the same Object will have to wait until first thread is done with the Object.

Example: This shows if more than one threads accessing getLine() method without synchronization.

```
// Example illustrates multiple threads are executing
// on the same Object at same time without synchronization.
import java.io.*;

class Line
{
    // if multiple threads(trains) will try to
    // access this unsynchronized method,
    // they all will get it. So there is chance
    // that Object's  state will be corrupted.
    public void getLine()
    {
        for (int i = 0; i < 3; i++)
        {
            System.out.println(i);
            try
            {
                Thread.sleep(400);
            }
            catch (Exception e)
            {
                System.out.println(e);
            }
        }
    }
}

class Train extends Thread
{
    // reference to Line's Object.
    Line line;

    Train(Line line)
    {
        this.line = line;
    }

    @Override
    public void run()
    {
        line.getLine();
    }
}

class GFG
{
    public static void main(String[] args)
    {
        // Object of Line class that is shared
        // among the threads.
```

```
        Line obj = new Line();

        // creating the threads that are
        // sharing the same Object.
        Train train1 = new Train(obj);
        Train train2 = new Train(obj);

        // threads start their execution.
        train1.start();
        train2.start();
    }
}
```

Output

```
0
0
1
1
2
2
```

There can be two trains (more than two) which need to use same at same time so there is chance of collision. Therefore to avoid collision we need to synchronize the line in which multiple want to run.

Example: Synchronized access to getLine() method on the same Object

```
// Example that shows multiple threads
// can execute the same method but in
// synchronized way.
class Line
{

    // if multiple threads(trains) trying to access
    // this synchronized method on the same Object
    // but only one thread will be able
    // to execute it at a time.
    synchronized public void getLine()
    {
        for (int i = 0; i < 3; i++)
        {
            System.out.println(i);
            try
```

```
                {
                    Thread.sleep(400);
                }
                catch (Exception e)
                {
                    System.out.println(e);
                }
            }
        }
    }

    class Train extends Thread
    {
        // Reference variable of type Line.
        Line line;

        Train(Line line)
        {
            this.line = line;
        }

        @Override
        public void run()
        {
            line.getLine();
        }
    }

    class GFG
    {
        public static void main(String[] args)
        {
            Line obj = new Line();

            // we are creating two threads which share
            // same Object.
            Train train1 = new Train(obj);
            Train train2 = new Train(obj);

            // both threads start executing .
            train1.start();
            train2.start();
        }
    }
```

Output:

```
0
1
2
```

```
0
1
2
```

Block Synchronization

If we only need to execute some subsequent lines of code not all lines (instructions) of code within a method, then we should synchronize only block of the code within which required instructions are exists.

For example, lets suppose there is a method that contains 100 lines of code but there are only 10 lines (one after one) of code which contain critical section of code i.e. these lines can modify (change) the Object's state. So we only need to synchronize these 10 lines of code method to avoid any modification in state of the Object and to ensure that other threads can execute rest of the lines within the same method without any interruption.

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

public class Geek
{
    String name = "";
    public int count = 0;

    public void geekName(String geek, List<String> list)
    {
        // Only one thread is permitted
        // to change geek's name at a time.
        synchronized(this)
        {
            name = geek;
            count++;  // how many threads change geek's name.
        }

        // All other threads are permitted
        // to add geek name into list.
        list.add(geek);
    }
}

class GFG
{
    public static void main (String[] args)
    {
        Geek gk = new Geek();
        List<String> list = new ArrayList<String>();
        gk.geekName("mohit", list);
        System.out.println(gk.name);
```

```
    }
}
```

Output :

```
mohit
```

Important points:

- When a thread enters into synchronized method or block, it acquires lock and once it completes its task and exits from the synchronized method, it releases the lock.
- When thread enters into synchronized instance method or block, it acquires Object level lock and when it enters into synchronized static method or block it acquires class level lock.
- Java synchronization will throw null pointer exception if Object used in synchronized block is null. For example, If in **synchronized(instance)** , **instance** is null then it will throw null pointer exception.
- In Java, **wait(), notify() and notifyAll()** are the important methods that are used in synchronization.
- You can not apply java **synchronized** keyword with the variables.
- Don’t synchronize on the non-final field on synchronized block because the reference to the non-final field may change anytime and then different threads might synchronize on different objects i.e. no synchronization at all.

Advantages

- **Multithreading:** Since java is multithreaded language, synchronization is a good way to achieve mutual exclusion on shared resource(s).
- **Instance and Static Methods:** Both synchronized instance methods and synchronized static methods can be executed concurrently because they are used to lock different Objects.

Limitations

- **Concurrency Limitations:** Java synchronization does not allow concurrent reads.
- **Decreases Efficiency:** Java synchronized method run very slowly and can degrade the performance, so you should synchronize the method when it is absolutely necessary otherwise not and to synchronize block only for critical section of the code.

This article is contributed by **Nitsdheerendra**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.