Java Synchronization

Synchronization is a process of handling resource accessibility by multiple thread requests. The main purpose of synchronization is to avoid thread interference. At times when more than one thread try to access a shared resource, we need to ensure that resource will be used by only one thread at a time. The process by which this is achieved is called synchronization. The synchronization keyword in java creates a block of code referred to as critical section.

General Syntax:

```
synchronized (object)
{
  //statement to be synchronized
}
```

Every Java object with a critical section of code gets a lock associated with the object. To enter critical section a thread need to obtain the corresponding object's lock.

Why we need Syncronization?

If we do not use syncronization, and let two or more threads access a shared resource at the same time, it will lead to distorted results.

Consider an example, Suppose we have two different threads **T1** and **T2**, T1 starts execution and save certain values in a file *temporary.txt* which will be used to calculate some result when T1 returns. Meanwhile, T2 starts and before T1 returns, T2 change the values saved by T1 in the file temporary.txt (temporary.txt is the shared resource). Now obviously T1 will return wrong result.

To prevent such problems, synchronization was introduced. With synchronization in above case, once T1 starts using *temporary.txt* file, this file will be **locked**(LOCK mode), and no other thread will be able to access or modify it until T1 returns.

Using Synchronized Methods

Using Synchronized methods is a way to accomplish synchronization. But lets first see what happens when we do not use synchronization in our program.

Example with no Synchronization

In this example, we are not using synchronization and creating multiple threads that are accessing display method and produce the random output.

```
class First
{
  public void display(String msg)
  {
    System.out.print ("["+msg);
    try
    {
      Thread.sleep(1000);
    }
    catch(InterruptedException e)
    {
      e.printStackTrace();
    }
    System.out.println ("]");
  }
}

class Second extends Thread
{
```

Output:

```
[welcome ] new [ programmer]
]
]
```

In the above program, object **fnew** of class First is shared by all the three running threads(ss, ss1 and ss2) to call the shared method(void **display**). Hence the result is nonsynchronized and such situation is called **Race condition**..

Synchronized Keyword

To synchronize above program, we must *synchronize* access to the shared **display()** method, making it available to only one thread at a time. This is done by using keyword **synchronized** with display() method.

```
synchronized void display (String msg)
```

Example : implementation of synchronized method

```
class First
{
  synchronized public void display(String msg)
  {
    System.out.print ("["+msg);
    try
    {
      Thread.sleep(1000);
    }
    catch(InterruptedException e)
    {
      e.printStackTrace();
    }
    System.out.println ("]");
  }
}

class Second extends Thread
{
```

Output:

```
[welcome]
[programmer]
[new]
```

Using Synchronized block

If want to synchronize access to an object of a class or only a part of a method to be synchronized then we can use synchronized block for it. It is capable to make any part of the object and method synchronized.

Example

In this example, we are using synchronized block that will make the display method available for single thread at a time.

```
class First
{
  public void display(String msg)
  {
    System.out.print ("["+msg);
    try
    {
      Thread.sleep(1000);
    }
    catch(InterruptedException e)
    {
      e.printStackTrace();
    }
    System.out.println ("]");
  }
}

class Second extends Thread
```

Output:

```
[welcome]
[new]
[programmer]
```

Because of synchronized block this program gives the expected output.

Difference between synchronized keyword and synchronized block

When we use synchronized keyword with a method, it acquires a lock in the object for the whole method. It means that no other thread can use any synchronized method until the current thread, which has invoked it's synchronized method, has finished its execution.

synchronized block acquires a lock in the object only between parentheses after the synchronized keyword. This means that no other thread can acquire a lock on the locked object until the synchronized block exits. But other threads can access the rest of the code of the method.

Which is more preferred - Synchronized method or Synchronized block?

In Java, synchronized keyword causes a performance cost. A synchronized method in Java is very slow and can degrade performance. So we must use synchronization keyword in java when it is necessary else, we should use Java synchronized block that is used for synchronizing critical section only.

← Prev
Next →