SNS COLLEGE OF ENGINEERING

Kurumbapalayam(Po), Coimbatore – 641 107
Accredited by NAAC-UGC with 'A' Grade
Approved by AICTE, Recognized by UGC & Affiliated to Anna University, Chennai

Department of Information Technology

19CS204 OBJECT ORIENTED PROGRAMMING

I YEAR /II SEMESTER

Topic – Thread Synchronization

Thread Synchronization

- Synchronization is a process **of handling resource accessibility by multiple thread requests**. The main purpose of synchronization is to **avoid thread interference&To prevent consistency problem.**

- When two or more threads need access to a shared resource, they need some way to ensure that the **resource will be used by only one thread at a time.**

- The process by which this is achieved is called synchronization.

- For example, If a thread is writing some data another thread may be reading the same data at that time. This may bring inconsistency.

- **Synchronization in java is the capability to control the access of multiple threads to any shared resource.**

Thread Synchronization

- Key to synchronization is the **concept of the monitor.**
- A monitor is **an object** that is used as a **mutually exclusive lock.**
- 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.
- These other threads are said to be **waiting for the monitor.**
- A thread that owns a monitor can reenter the same monitor if it so desires.

Thread Synchronization

- Synchronization can be accomplished by two ways in java,
  - By Synchronized Method
  - By Synchronized Statement or Block

Synchronized Method

- To enter an **object's monitor**, just call a method that has been **modified with the synchronized keyword.**
- **While a thread is inside a synchronized method**, all other threads that try to call it (or any other synchronized method) on the same instance have to wait.
- To exit the monitor and relinquish control of the object to the next waiting thread, the owner of the monitor simply returns from the synchronized method.

Thread Synchronization

Example without synchronization

```
class Table{
void printTable(int n){//method not synchronized
   for(int i=1;i<=5;i++){
     System.out.println(n*i);
     try{
      Thread.sleep(400);
     }catch(Exception e){System.out.println(e);}
   }  }
}
class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
} }
```

```
class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
this.t=t;
}
public void run(){
t.printTable(100);
}
}
public class TestSynchronization1{
public static void main(String args[]){
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
}
}
```

Thread Synchronization

Example 1 with Synchronized Method

```
class Table{
synchronized void printTable(int n){
   for(int i=1;i<=5;i++){
     System.out.println(n*i);
     try{
      Thread.sleep(400);
     }catch(Exception e){System.out.println(e);}
   }  }
}
class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
} }
class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
this.t=t;
}
public void run(){
t.printTable(100);
}
}
public class TestSynchronization1{
public static void main(String args[]){
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
}
}
```

Thread Synchronization

Synchronized block

- While creating synchronized methods within classes that you create is an easy and effective means of achieving synchronization, it will not work in all cases.

- To understand why, consider the following. Imagine that you want to synchronize access to objects of a class that was not designed for multithreaded access.

- That is, the class does not use synchronized methods.

- This is the general form of the synchronized statement:

```
synchronized(objRef) {
// statements to be synchronized
}
```

- Here, objRef is a reference to the object being synchronized.

- A synchronized block ensures that a call to a synchronized method that is a member of objRef’s class occurs only after the current thread has successfully entered objRef’s monitor.

Thread Synchronization

Example 2 Synchronized block

```
class Table{
 void printTable(int n){
  synchronized(this){//synchronized block
for(int i=1;i<=5;i++){
 System.out.println(n*i);
 try{
Thread.sleep(400);
 }catch(Exception e){System.out.println(e);}
}   }
}//end of the method
}
class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
}   }
```

```
class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
this.t=t;
}
public void run(){
t.printTable(100);
}
}
public class TestSynchronizedBlock1{
public static void main(String args[]){
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
}
}
```

Thread Synchronization

Example 3 Synchronized Method

```
class Callme {
synchronized void call(String msg) {
System.out.print("[" + msg);
try {
Thread.sleep(1000);
} catch(InterruptedException e) {
System.out.println("Interrupted");
}
System.out.println("]");}}
class Caller implements Runnable {
String msg;
Callme target;
Thread t;
public Caller(Callme targ, String s) {
target = targ;
msg = s;
t = new Thread(this);
t.start();
}
```

```
public void run() {
target.call(msg);
}
}

public class Synch {
public static void main(String args[]) {
Callme target = new Callme();
Caller ob1 = new Caller(target, "Hello");
Caller ob2 = new Caller(target, "Synchronized");
Caller ob3 = new Caller(target, "World");
}
}
```

Thread Synchronization

Example 4 Synchronized Block

```
class Callme {
void call(String msg) {
System.out.print("[" + msg);
try {
Thread.sleep(1000);
} catch(InterruptedException e) {
System.out.println("Interrupted");
}
System.out.println("]");}}
class Caller implements Runnable {
String msg;
Callme target;
Thread t;
public Caller(Callme targ, String s) {
target = targ;
msg = s;
t = new Thread(this);
t.start();
}
public void run() {
target.call(msg);
}}
```

```
public void run() {
synchronized (target) {
target.call(msg);
}
}
}

public class Synch {
public static void main(String args[]) {
Callme target = new Callme();
Caller ob1 = new Caller(target, "Hello");
Caller ob2 = new Caller(target, "Synchronized");
Caller ob3 = new Caller(target, "World");
}
}
```

THANK YOU