**Synchronized method**

```
class Table{
synchronized void printTable(int n)
//synchronized method
{
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 TestSynchronization2{
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();
}
}
```

**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();
}}
```

This code demonstrates the concept of synchronization in Java, where two threads (MyThread1 and MyThread2) attempt to print multiplication tables for 5 and 100, respectively, using the same shared object Table. Since the printTable method is synchronized, only one thread can execute this method at a time on the same object, ensuring that the multiplication tables do not overlap or cause inconsistency.

**Synchronized Method**: The printTable(int n) method is marked synchronized. This ensures that only one thread can execute the method on the same object at any given time. This is useful to avoid issues like race conditions or corrupted data when multiple threads access shared resources.

**Thread Classes:** MyThread1 and MyThread2 extend Thread and override the run() method. Each thread calls the synchronized printTable method on the same Table object (obj), passing in different values (5 and 100).

**Thread Execution:** t1.start() and t2.start() will start both threads concurrently. Because the printTable method is synchronized, one thread will wait for the other to finish before it can start executing the method.

**Expected Output:**

5
10
15
20
25
100
200
300
400
500

**If the synchronized keyword is not used**, both threads (MyThread1 and MyThread2) will be able to execute the printTable method concurrently on the same object (Table). This can lead to interleaved output, where the multiplication tables for 5 and 100 may get mixed up because both threads might try to print at the same time.

**Example Output (Without Synchronization):**

5
100
200
10
300
15
20
400
25
500
30
35
40
45
50

**Unpredictable Behavior:** The exact interleaving of numbers depends on the thread scheduling done by the Java Virtual Machine (JVM) and the underlying operating system. Each time you run the program, the output could change because thread execution order is not guaranteed.