What is Race condition?

“The unpredictable behavior of the program which is caused by 2 or more threads which access and modify the shared variables is called Race Condition”.

Consider the following example.

A screenshot of a social media post

Description automatically generated

The DoWork() is being called twice, once via a child thread and once by main method(main thread). We expect \* is printed 10 times (5times each by each by each of these 2 threads).

But the counter variable ‘i’ is a shared variable across these 2 threads, which will be accessed and modified by both threads asynchronously. So it may print 5 \* or 6 \* etc… i.e. the behavior becomes un predictable. This is the example of Race condition.

The solution to the Race Condition is to use locking around the shared variables, which will create a critical region and lets only one thread at a time to modify the shared resources.

Using a private reference type (e.g. Object type) is commonly used to create the lock\ critical region.

Ex: private static Object lockHandle = new Object();

lock(lockHandle)

{

I = I +1;

}

Lock is just a syntactic sugar of Monitor object (used in creating the critical regions).

Instead of lock you can also use

Try

{

Monitor.Enter(lockHandle) //obtain the lock

{

I = I +1;

}

}

Finally

{

Monitor.Exit(lockHandle); //release the lock

}

Another variant of Monitor will provide the time out option for other threads attempting to enter the critical region, if they have wait for a long time.

Monitor.TryEnter()

A screenshot of a social media post

Description automatically generated

Thread Synchronization:

This will establish the robust communication between 2 threads while they are accessing the shared resources.

A screen shot of a blackboard

Description automatically generated

A screen shot of a blackboard

Description automatically generated

A close up of a blackboard

Description automatically generated

A close up of a blackboard

Description automatically generated

