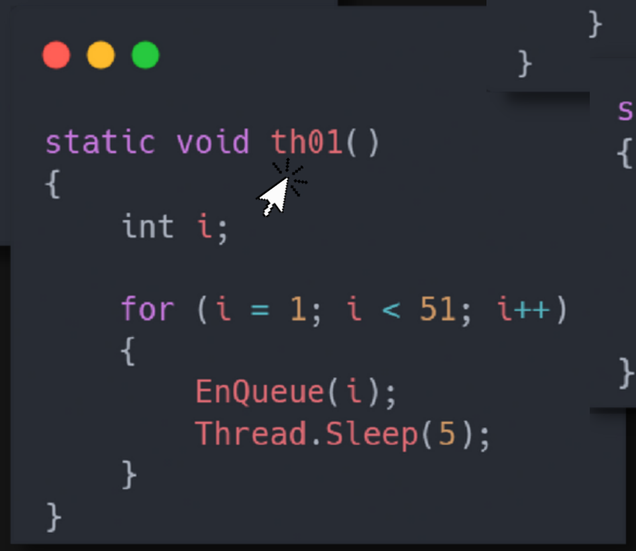
Operating Systems

Case Study #2

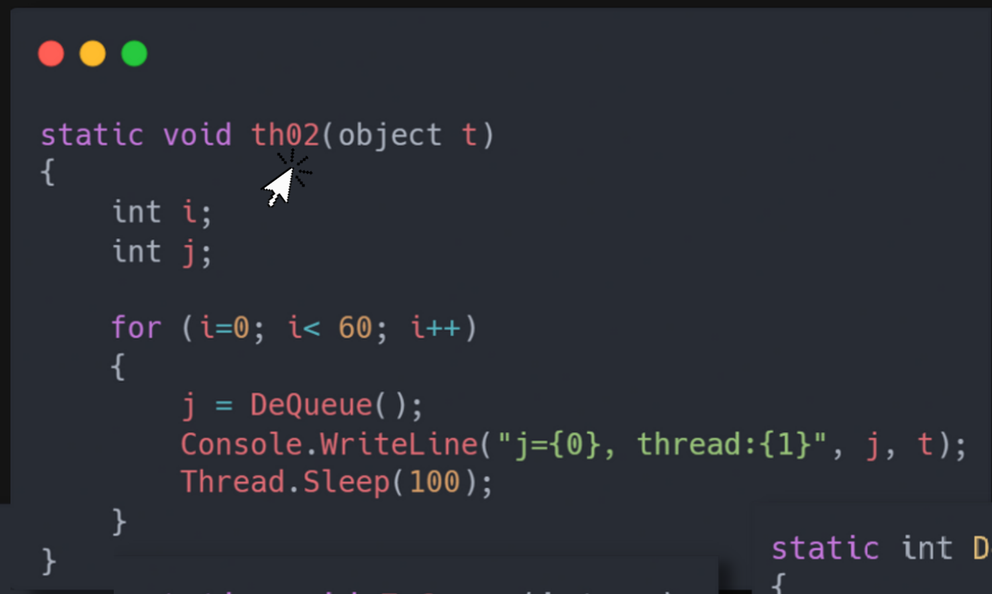
Group

Version 0

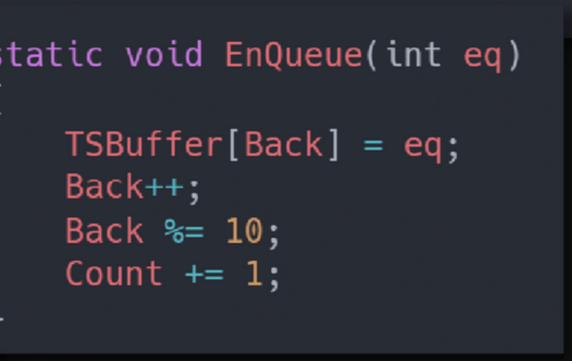
```
static void Main(string[] args)
{
    Thread t1 = new Thread(th01);
    //Thread t11 = new Thread(th01);
    Thread t2 = new Thread(th02);
    //Thread t21 = new Thread(th02);
    //Thread t22 = new Thread(th02);

    t1.Start();
    //t11.Start();
    t2.Start(1);
    //t21.Start(2);
    //t22.Start(3);
}
```



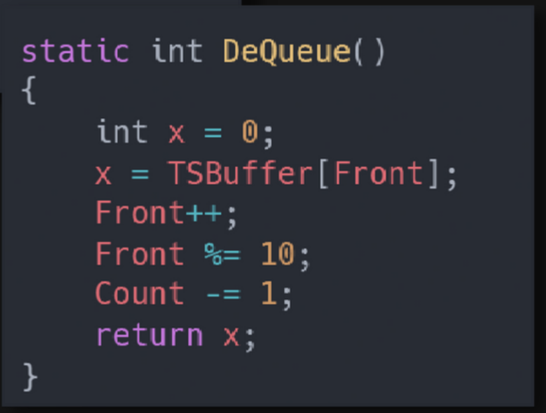
```
static void th01()
{
    int i;

    for (i = 1; i < 51; i++)
    {
        EnQueue(i);
        Thread.Sleep(5);
    }
}
```



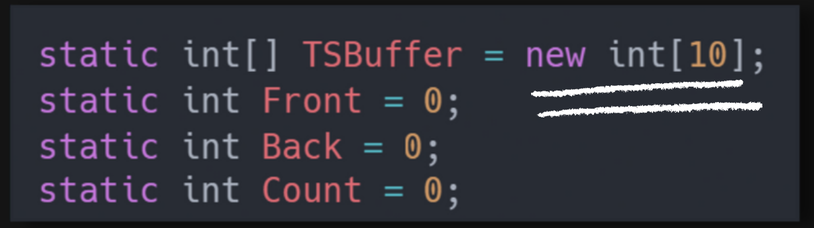
```
static void th02(object t)
{
    int i;
    int j;

    for (i=0; i< 60; i++)
    {
        j = DeQueue();
        Console.WriteLine("j={0}, thread:{1}", j, t);
        Thread.Sleep(100);
    }
}
```



```
static void EnQueue(int eq)
{
    TSBuffer[Back] = eq;
    Back++;
    Back %= 10;
    Count += 1;
}
```

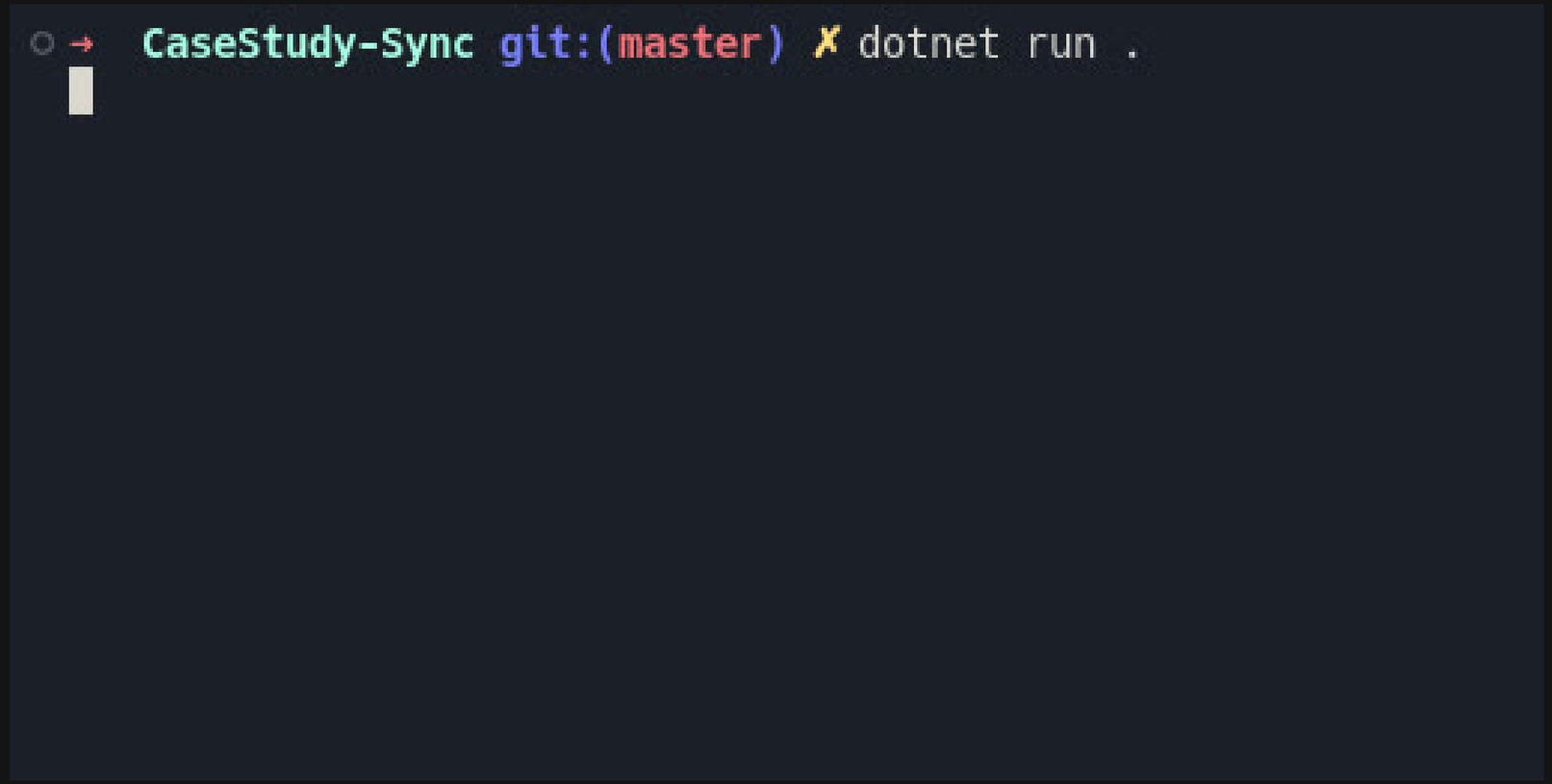


```
static int DeQueue()
{
    int x = 0;
    x = TSBuffer[Front];
    Front++;
    Front %= 10;
    Count -= 1;
    return x;
}
```

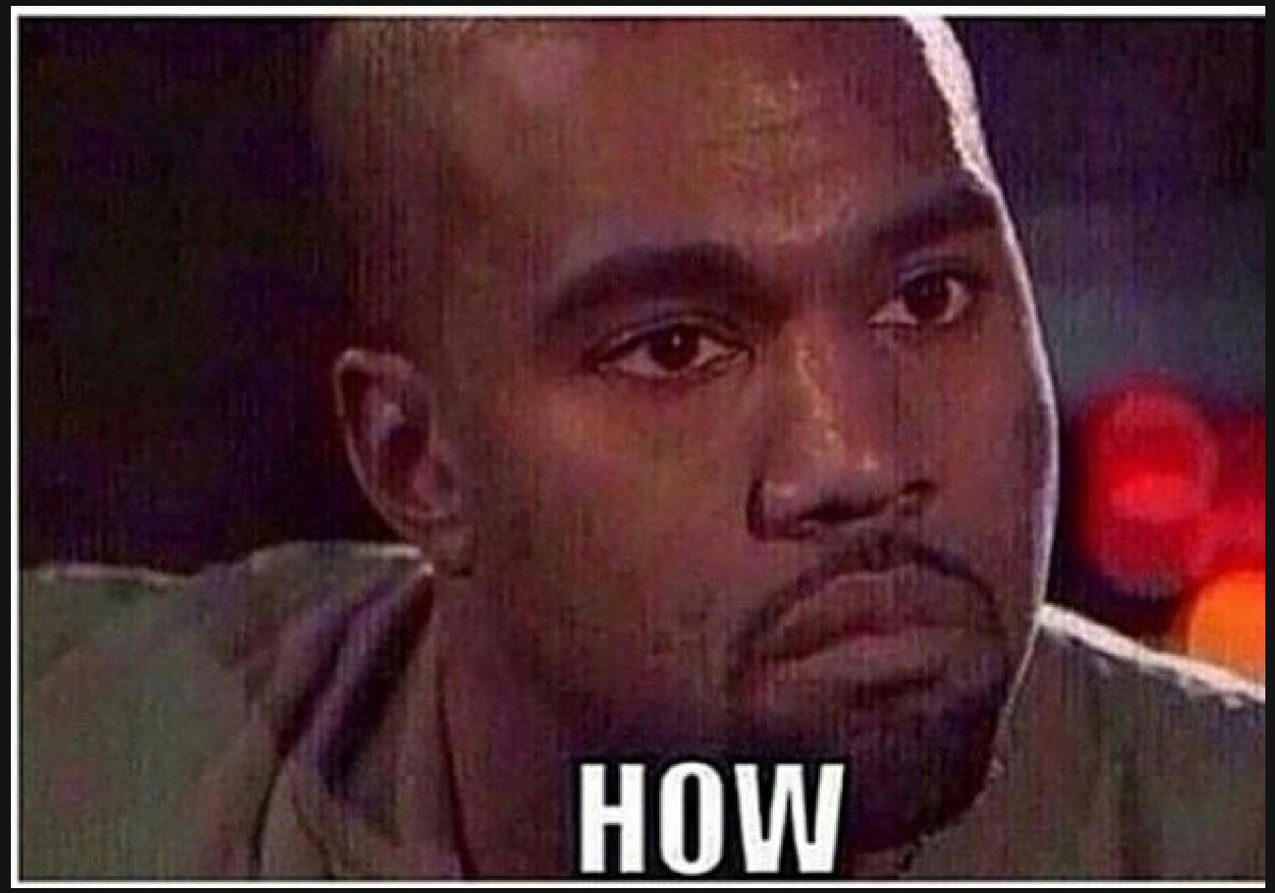


```
static int[] TSBuffer = new int[10];
static int Front = 0;
static int Back = 0;
static int Count = 0;
```

Result:



What is the problems?




```
static void th01()
{
    int i;

    for (i = 1; i < 51; i++)
    {
        EnQueue(i);
        Thread.Sleep(5);
    }
}
```

```
static void th02(object t)
{
    int i;
    int j;

    for (i=0; i< 60; i++)
    {
        j = DeQueue();
        Console.WriteLine("j={0}, thread:{1}", j, t);
        Thread.Sleep(100);
    }
}
```

สีหน้าของพวกเราเมื่อเห็นโจทย์

Thread safety is a computer programming concept applicable to multi-threaded code.

- wikipedia

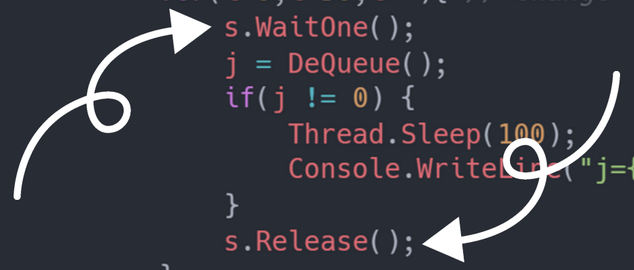
- ให้แก้ไขดัดแปลงโปรแกรมที่กำหนดให้ให้มีคุณสมบัติดังนี้
  - ทำงานได้อย่างถูกต้อง
  - ทำงานกับ Thread ได้อย่างถูกต้อง (Thread safe)

Version 1

```
private static Semaphore s = new Semaphore(1, 1);
```

```
static void th01()
{
    for(int i=1;i<51;i++){
        s.WaitOne();
        EnQueue(i);
        Thread.Sleep(5);
        s.Release();
    }
}
```

```
static void th02(object t)
        {
            int i,j;

            for(i=0;i<50;i++){ // change this to 50, 'cause why not
                s.WaitOne();
                j = DeQueue();
                if(j != 0) {
                    Thread.Sleep(100);
                    Console.WriteLine("j={0}, thread:{1}", j, t);
                }
                s.Release();
            }
        }
```



```
static void Main(string[] args)
{
    Thread t1 = new Thread(th01);
    Thread t2 = new Thread(th02);

    t1.Start();
    t2.Start(1);
}
```

Result:

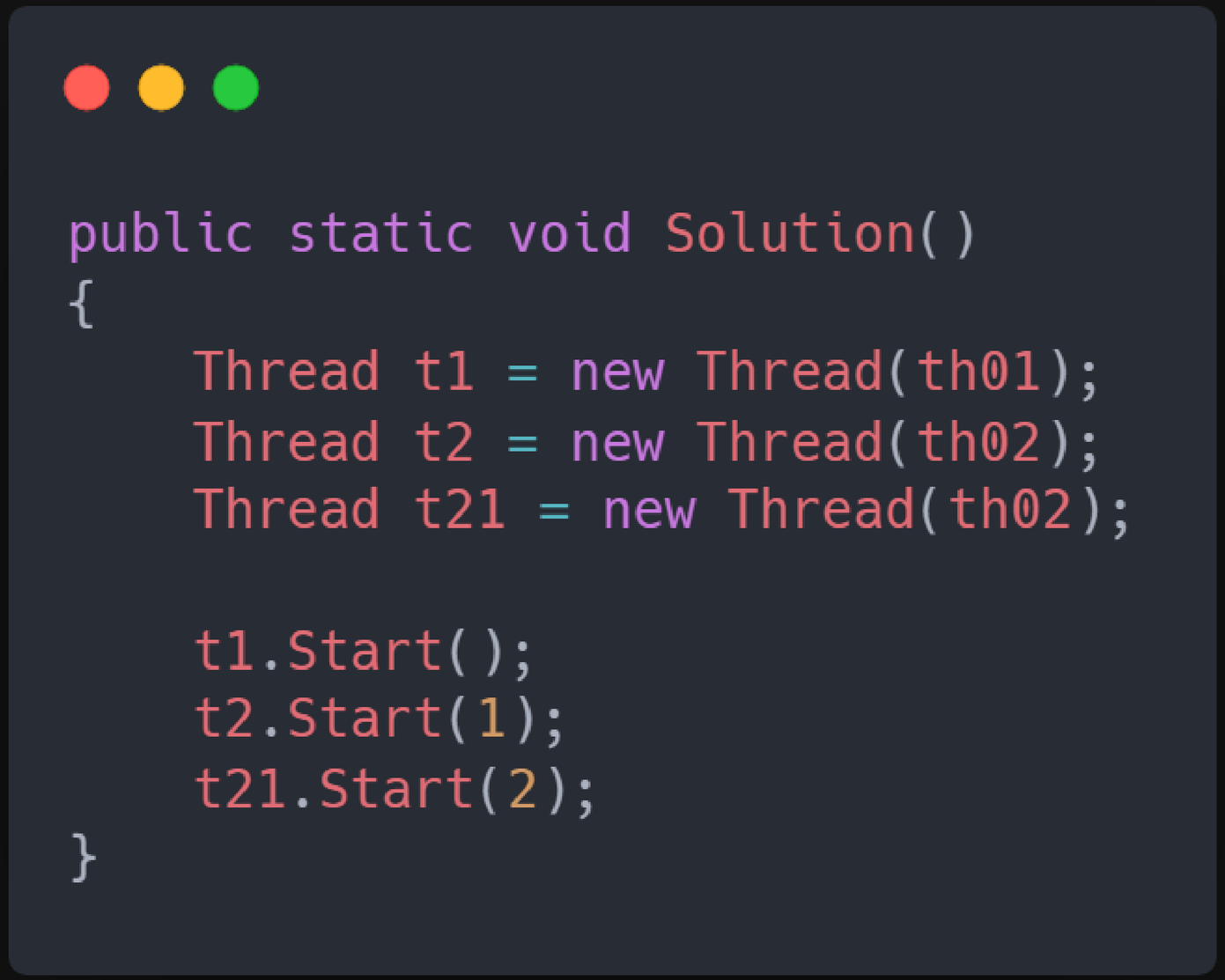
```
→  CaseStudy-Sync git:(master) ✗ dotnet run .
Version 1
j=1, thread:1
j=2, thread:1
j=3, thread:1
j=4, thread:1
j=5, thread:1
j=6, thread:1
j=7, thread:1
j=8, thread:1
j=9, thread:1
j=10, thread:1
j=11, thread:1
j=12, thread:1
j=13, thread:1
```

Version 2

```
private static int exitflag = 0;
```

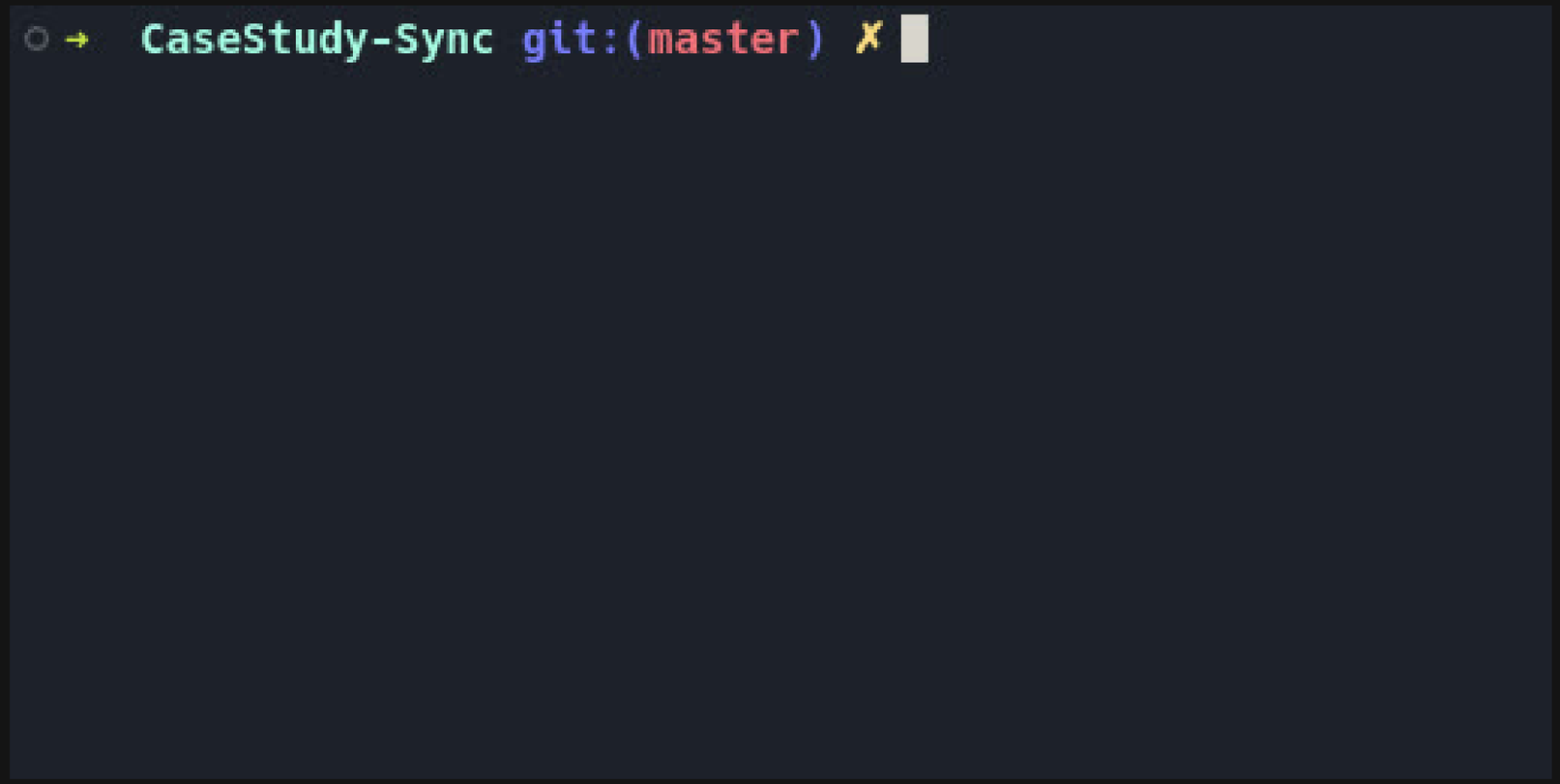
```
static void th01()
{
    for(int j=0;j<5;j++) {
        s.WaitOne();
        for (int i = (10*j)+1; i < ((j+1)*10)+1; i++)
        {
            EnQueue(i);
            Thread.Sleep(5);
        }
        s.Release();
    }
    exitflag = 1;
```

```
static void th02(object t)
{
    int i,j;
    while(exitflag == 0) {
        s.WaitOne();
        for(i=0;i<5;i++){
            j = DeQueue();
            if(j != 0) {
                Thread.Sleep(100);
                Console.WriteLine("j={0}, thread:{1}", j, t);
            }
        }
        s.Release();
    }
}
```



```
public static void Solution()
{
    Thread t1 = new Thread(th01);
    Thread t2 = new Thread(th02);
    Thread t21 = new Thread(th02);

    t1.Start();
    t2.Start(1);
    t21.Start(2);
}
```

Result:

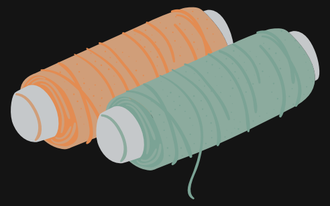


Version 3

```
static private int BUFFER_SIZE = 10;
static int[] TSBuffer = new int[BUFFER_SIZE];
static int Front = 0;
static int Back = 0;
static int Count = 0;

private static Semaphore s = new Semaphore(1, 1);

private static int exitflag = 0;

private static bool canEnqueue = true;
```

```
static void EnQueue(int eq)
{
    TSBuffer[Back] = eq;
    Back++;
    Back %= 10;
    Count += 1;
}
```

```
static void th01()
{
    int j = 0;
    while(j < 5 && exitflag == 0){
        if(canEnqueue){
            s.WaitOne();
            for (int i = (10*j)+1; i < ((j+1)*10)+1; i++)
            {
                EnQueue(i);
                Thread.Sleep(5);
            }
            canEnqueue = false;
            j+=1;
            s.Release();
        }
    }
}
```

Thread enqueue  [1,2,3,4,5,6,7,8,9,10] canenqueue = false

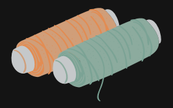
Dequeue [1,2,3,4,5,6,7,8,9,10] → [ ]

Thread 1 Thread 2 Thread 3 Thread 1 Thread 2 Thread 3 Thread 1 Thread 2 Thread 3 Thread 1

```
static int DeQueue()
{
    int x = 0;
    x = TSBuffer[Front];
    Front++;
    if(Front>=BUFFER_SIZE){
        canEnqueue = true;
    }
    Front %= 10;
    Count -= 1;
    return x;
}
```

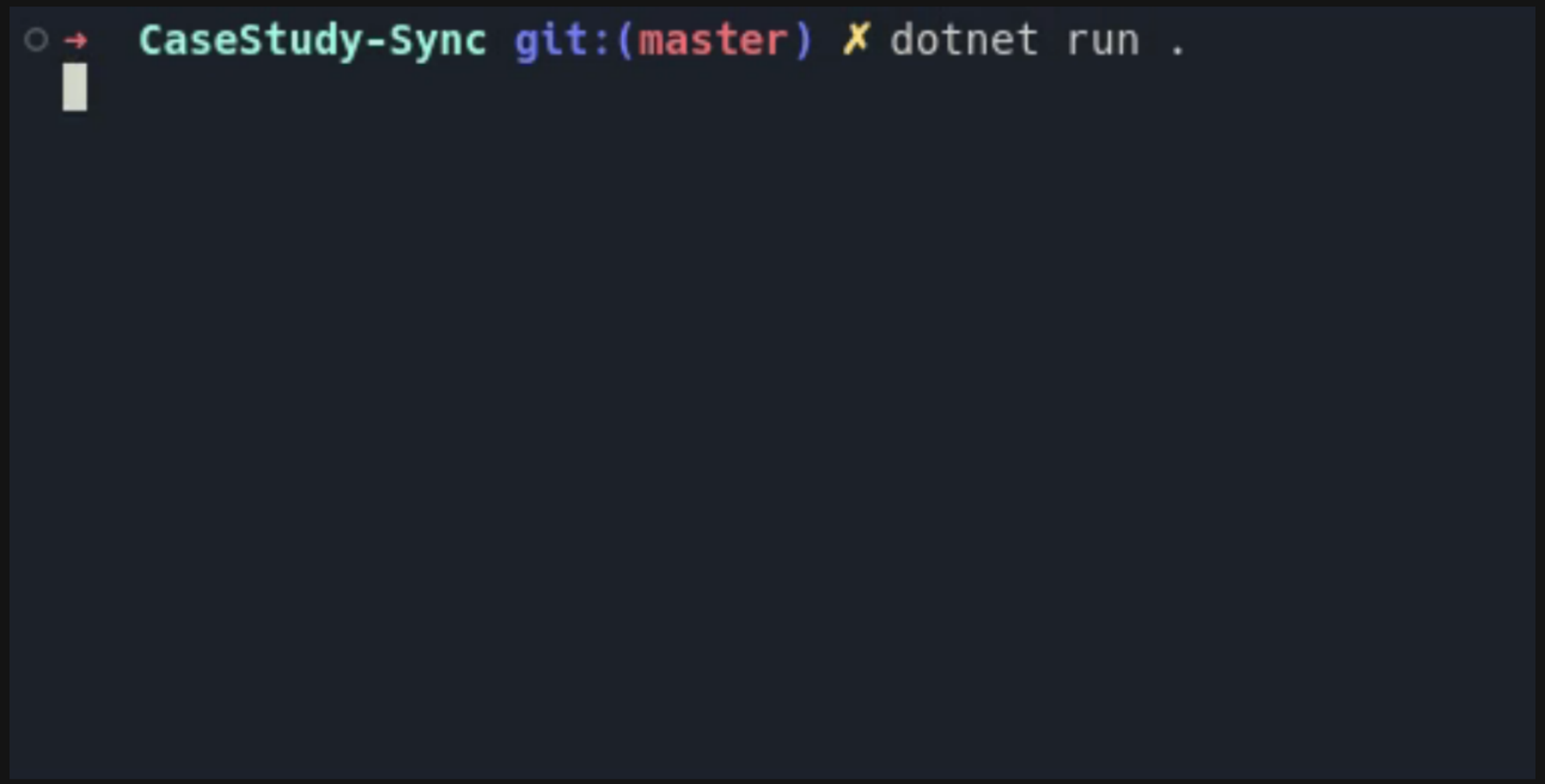
canenqueue = True

```
static void th02(object t)
{
    int j;
    while(exitflag == 0) {
        s.WaitOne();
            if(!canEnqueue){
                j = DeQueue();
                Thread.Sleep(100);
                Console.WriteLine("j={0}, thread:{1}", j, t);
                if (j == 50){
                    exitflag = 1;
                }
            }
        s.Release();
    }
}
```

Thread enqueue  [11,12,13,14,15,16,17,18,19,20]

```
public static void Solution()
{
    Thread t1 = new Thread(th01);
    Thread t11 = new Thread(th011);
    Thread t2 = new Thread(th02);
    Thread t21 = new Thread(th02);
    Thread t22 = new Thread(th02);

    t1.Start();
    // t11.Start();
    t2.Start(1);
    t21.Start(2);
    t22.Start(3);
}
```

Result:

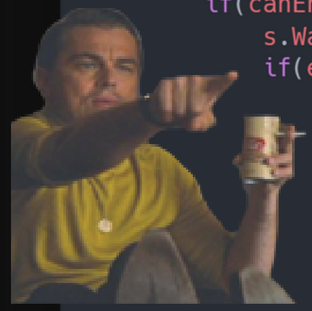



นักศึกษาที่พยายามบังคับให้
Thread ทำงาน sync กัน
น้อง Thread

Version 4

```
static void th01()
{
    int j = 0;
    while(j < 5 && exitflag == 0){
        if(canEnqueue){
            s.WaitOne();
            for (int i = (10*j)+1; i < ((j+1)*10)+1; i++)
            {
                EnQueue(i);
                Thread.Sleep(5);
            }
            canEnqueue = false;
            j+=1;
            s.Release();
        }
    }
}
```

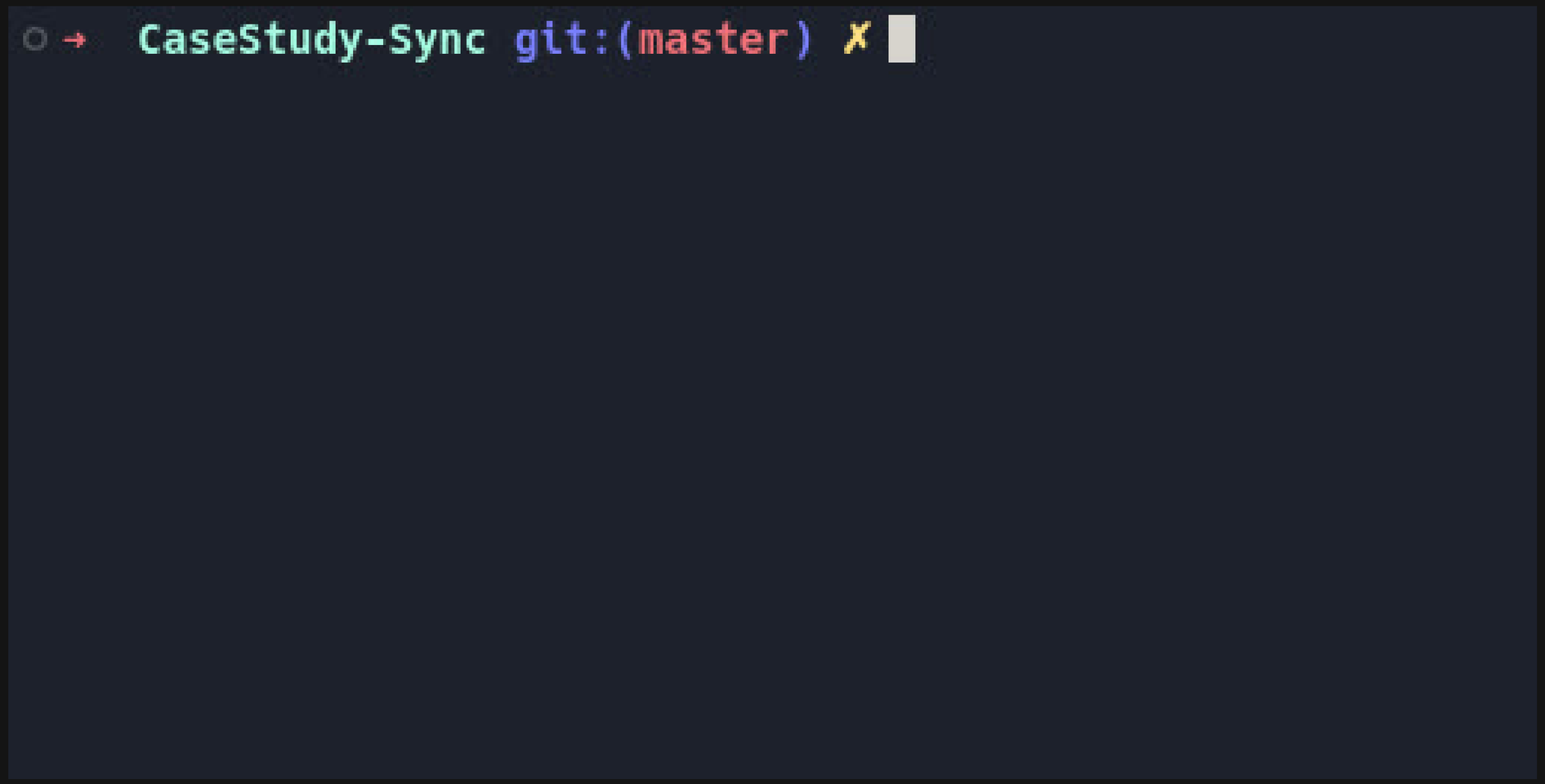
```
static void th011()
{
    int j = 0;
    while(j < 5){
        if(canEnqueue){
            s.WaitOne();
            if(exitflag == 1){
                Back = 9;
                for (int i = (10*j)+100; i < ((j+1)*10)+101; i++)
                {
                    EnQueue(i);
                    Thread.Sleep(5);
                }
                canEnqueue = false;
                j+=1;
            }
            s.Release();
        }
    }
}
```



```
static void th02(object t)
{
    int j;
    while(exitflag != 2) {
        s.WaitOne();
            if(!canEnqueue){
                j = DeQueue();
                if(j != 0) {
                    Thread.Sleep(100);
                    Console.WriteLine("j={0}, thread:{1}", j, t);
                }
                if (j == 50){
                    exitflag = 1;
                }
                if (j == 150) {
                    exitflag = 2;
                }
            }
        s.Release();
    }
}
```

```
public static void Solution()
{
    Thread t1 = new Thread(th01);
    Thread t11 = new Thread(th011);
    Thread t2 = new Thread(th02);
    Thread t21 = new Thread(th02);
    Thread t22 = new Thread(th02);

    t1.Start();
    t11.Start();
    t2.Start(1);
    t21.Start(2);
    t22.Start(3);
}
```

Result:



Version 5





GFXNITROZ

```
private static Object _Lock = new Object();
private static bool enqueueDone1 = false;
private static bool enqueueDone2 = false;
```

```
static void EnQueue(int eq){
  TSBuffer[Back] = eq;
  Back++;
  Back %= 10;
  Count += 1;
}
```

```
static void th01()
{
    int i = 1;
    while(!enqueueDone1) {
        lock (_Lock) {
            if(Count == 0 && canEnqueue) {
                EnQueue(i);
                Thread.Sleep(5);
                i++;
            }
        }
        if(i == 51) enqueueDone1 = true;
    }
}
```

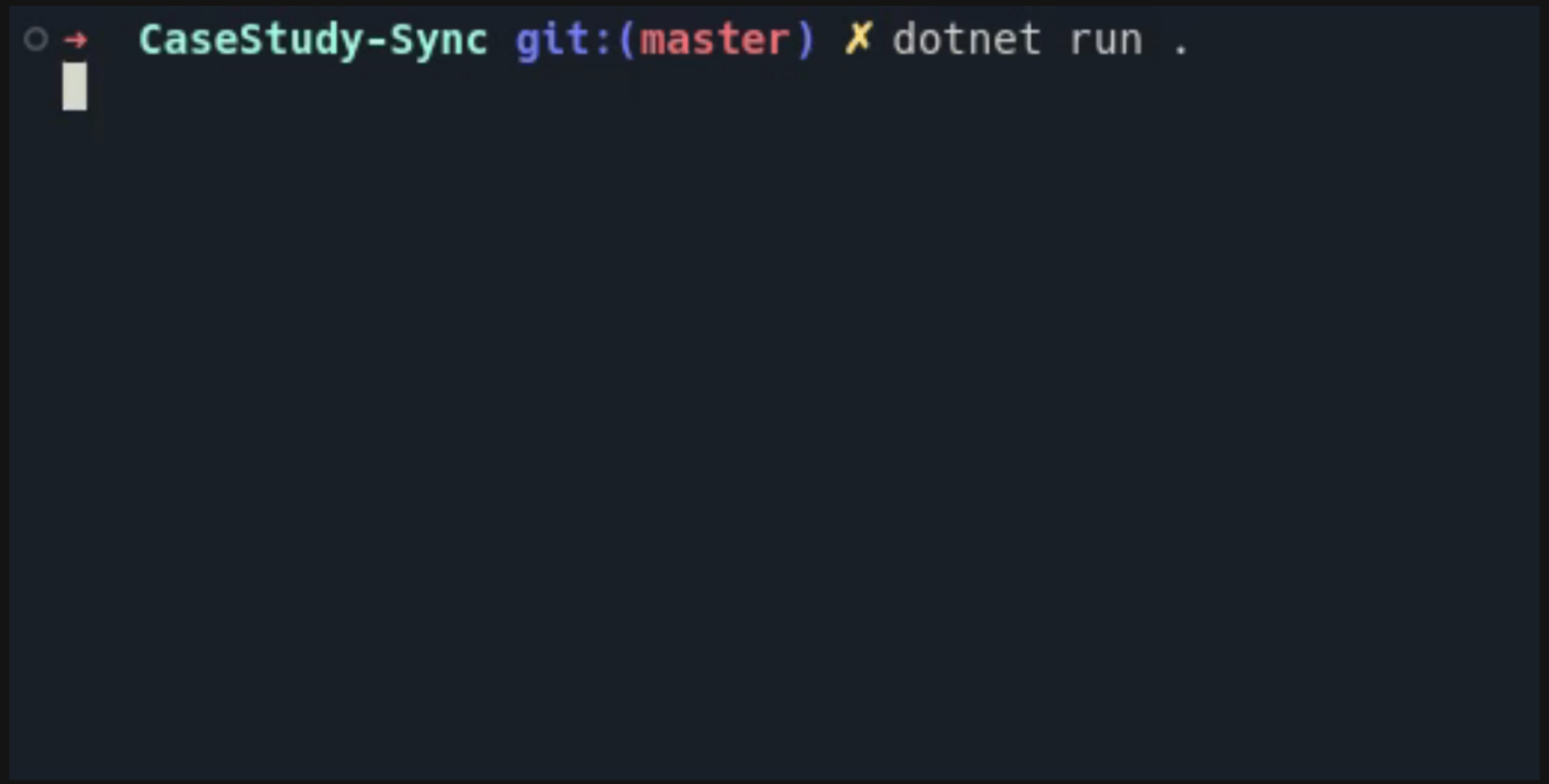
```
static void th011()
{
    int i = 101;
    while(!enqueueDone2) {
        lock (_Lock) {
            if(Count == 0 && canEnqueue) {
                EnQueue(i);
                Thread.Sleep(5);
                i++;
            }
        }
        if(i==151) enqueueDone2 = true;
    }
}
```

```
static void th02(object t)
{
    int j;
    while(exitflag == 0) {
        lock (_Lock) {
            if(Count > 0) {
                j = DeQueue();
                if(j != 0) {
                    Thread.Sleep(100);
                    Console.WriteLine("j={0}, thread:{1}", j, t);
                }
                if (enqueueDone1 && enqueueDone2) {
                    exitflag = 1;
                }
            }
        }
    }
}
```

```
static int DeQueue(){
  int x = 0;
  x = TSBuffer[Front];
  Front++;
  Front %= 10;
  Count -= 1;
  return x;
}
```

```
public static void Solution()
{
    Thread t1 = new Thread(th01);
    Thread t11 = new Thread(th011);
    Thread t2 = new Thread(th02);
    Thread t21 = new Thread(th02);
    Thread t22 = new Thread(th02);

    t1.Start();
    t11.Start();
    t2.Start(1);
    t21.Start(2);
    t22.Start(3);
}
```

Result:



Summary

Objective

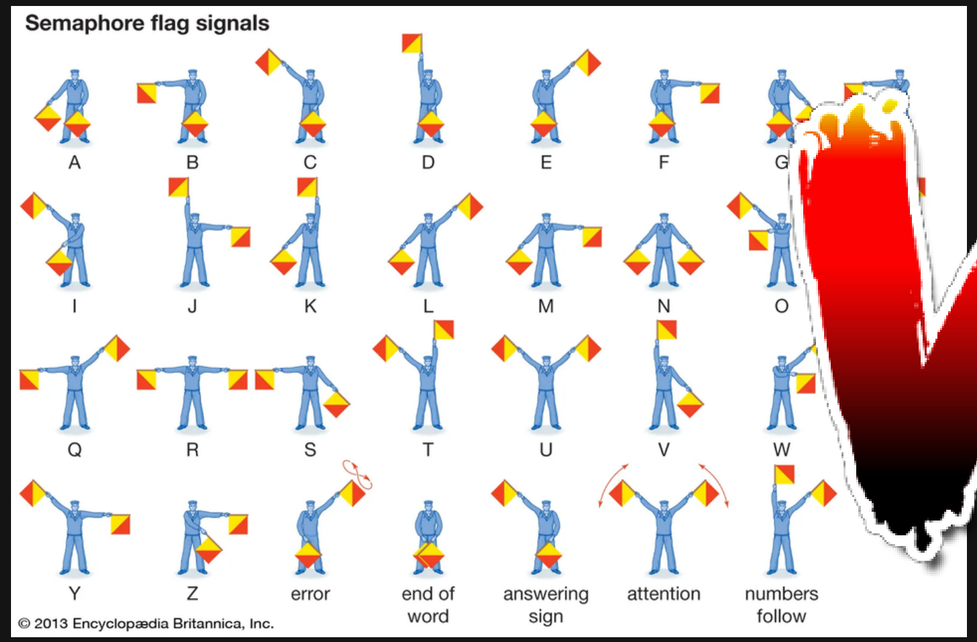
- ให้แก้ไขดัดแปลงโปรแกรมที่กำหนดให้ให้มีคุณสมบัติดังนี้
  - ทำงานได้อย่างถูกต้อง
  - ทำงานกับ Thread ได้อย่างถูกต้อง (Thread safe)

Thread safety is a computer programming concept applicable to multi-threaded code. Thread-safe code only manipulates shared data structures in a manner that ensures that all threads behave properly and fulfill their design specifications without unintended interaction. There are various strategies for making thread-safe data structures.[1][2]

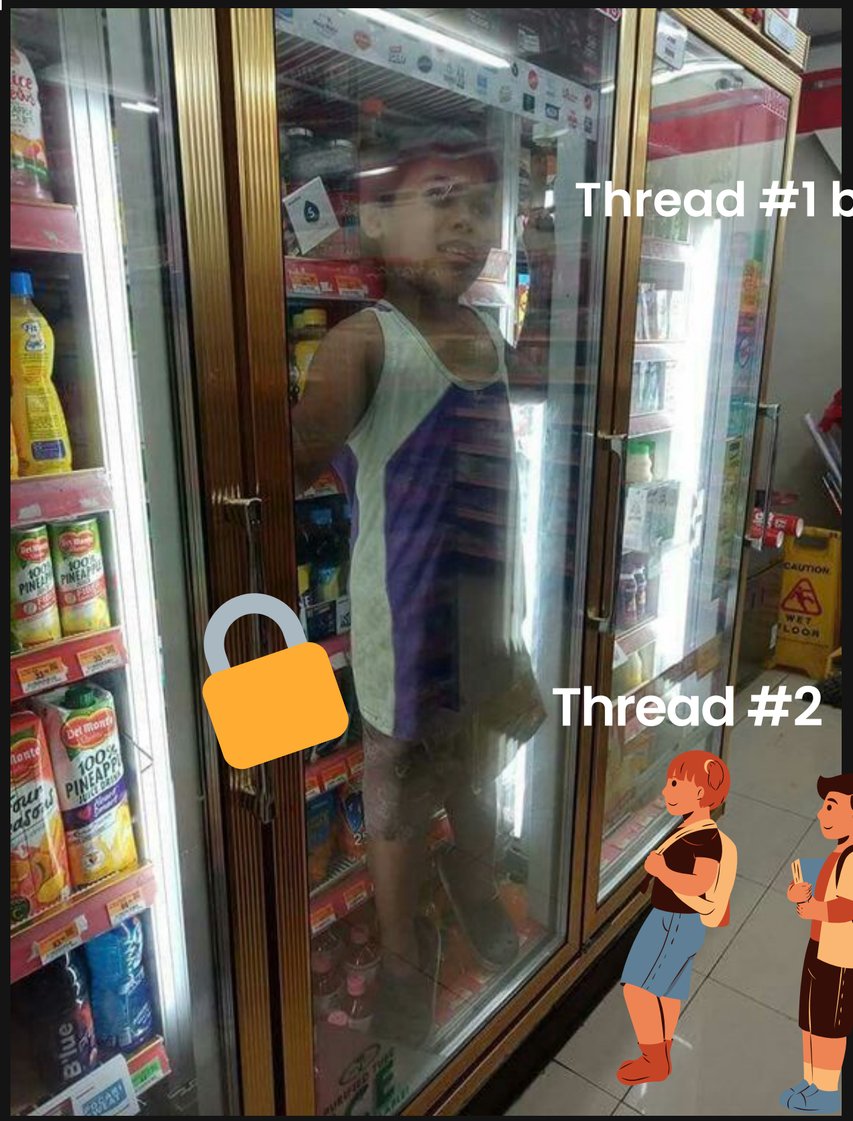
– wikipedia

Semaphore

Lock

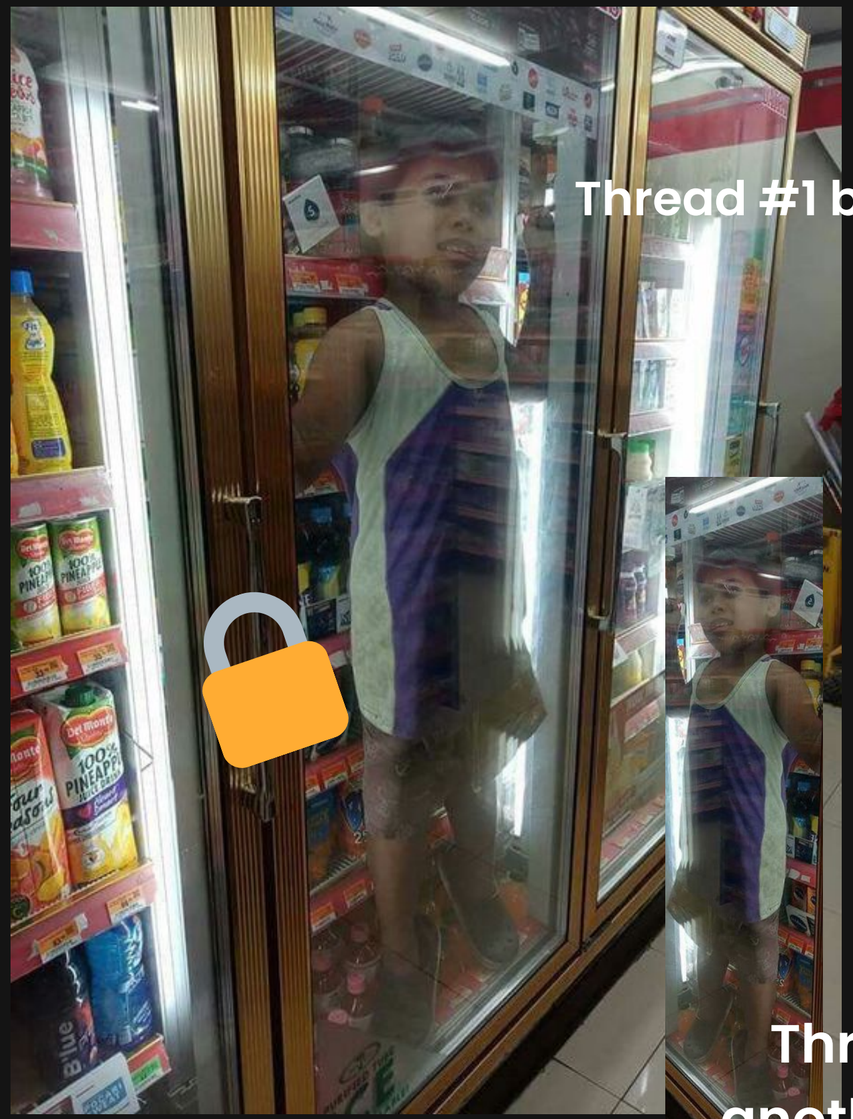


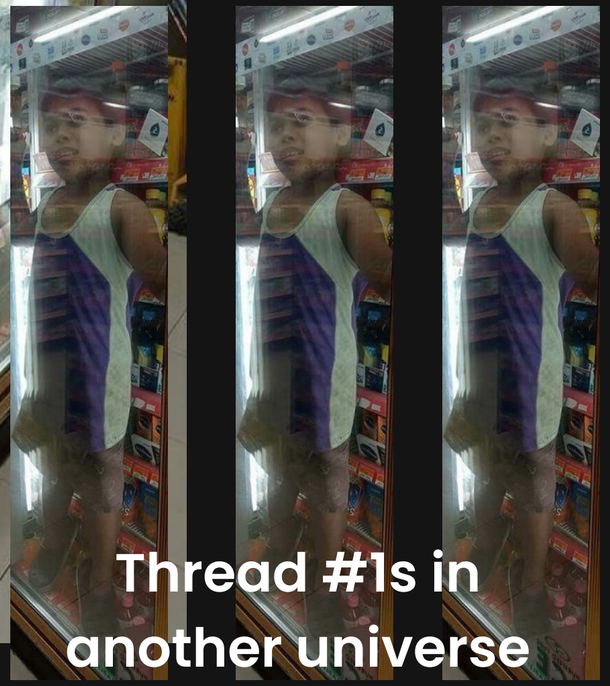
Semaphore



Lock


scheduler
Thread #1
Thread #2
Thread #3


Thread #1 be like


Thread #1s in another universe