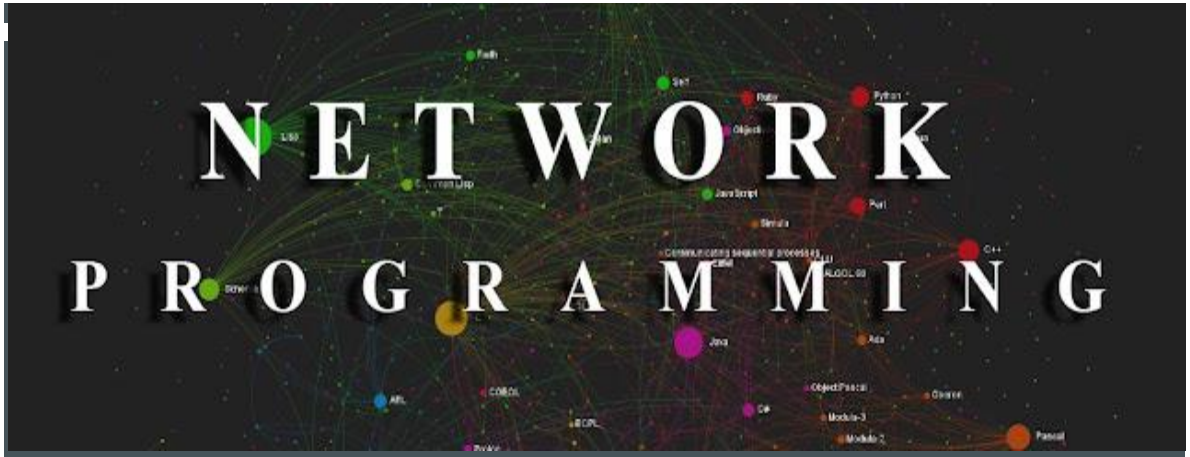


IT432 - NETWORK PROGRAMMING

DR. ALI HUSSEIN AHMED

ALI.HUSSEIN@AUN.EDU.EG

IT DEPT - OFFICE NO. 312

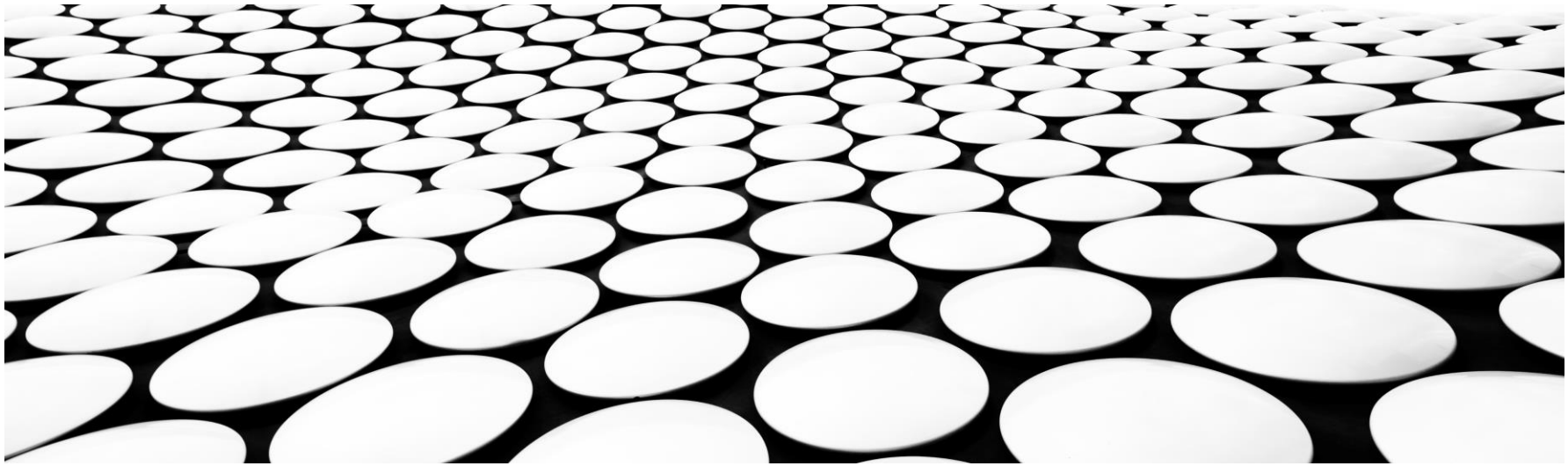
2022

جامعة أسيوط

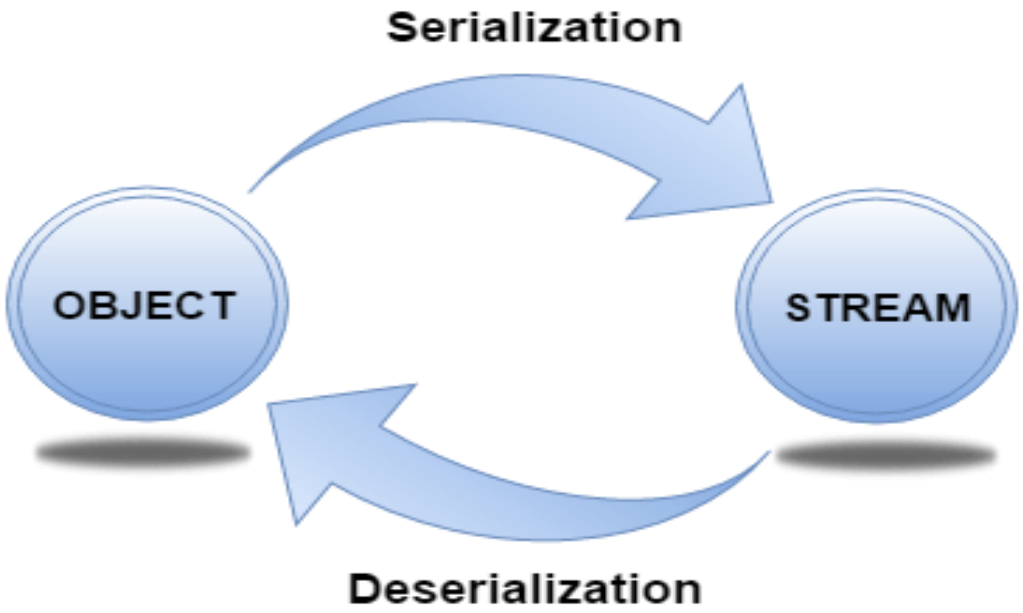
كلية الحاسبات والمعلومات

كلية معتمدة من الهيئة القومية لضمان الجودة والاعتماد

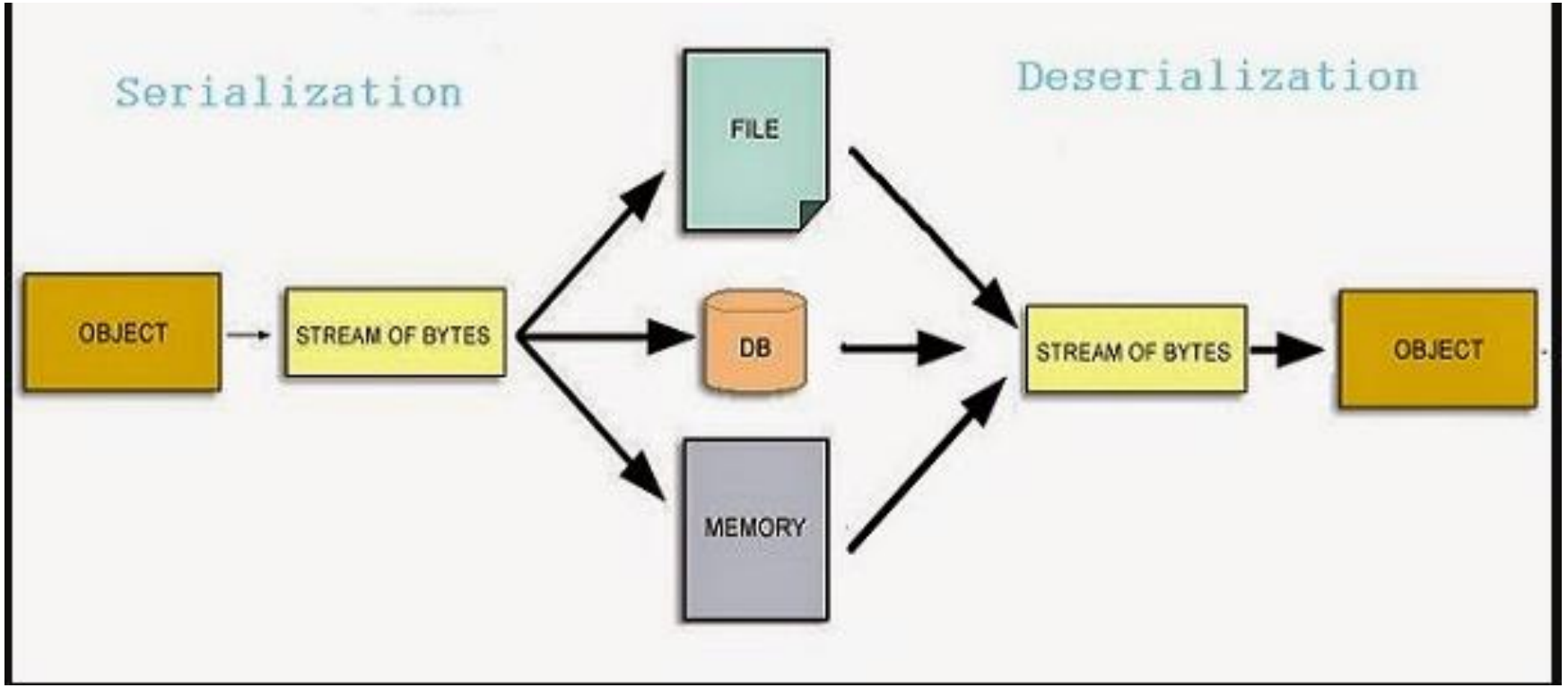
LECTURE 2



Serialization



What is object serialization ?



Serialization

- Serialization
  - Writes an object to a file or any type of streams as a series of bytes.
- Deserialization
  - Reads a serialized file in order to reestablish or reconstruct the serialized object
- Serialization and deserialization is supported via classes that implement the *Iformatter* interface:
  - **BinaryFormatter** and **SoapFormatter**
- Methods in **Iformatter**:
  - **Serialize** and **Deserialize**
- **During this course we will study only BinaryFormatter for binary object serialization**

WHY SERIALIZATION ?

- Serialization is the best way to save objects in streams so that we can “send” it any where or even save it immediately.
- Object state can be resorted after deserialization
- Privacy and object state issues !!!
  - What did you think the problem of object state?

Example of Serialization in C#

- In this example we will serialize and restore a *person* object compositing a *date* object.
- We will define class Date and Person.
- A controller program will make instance from the person class and serialize it to a file and then restore/deserialize it back again into memory

THE USED NAMESPACES

```
using System;
using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
```

STEP 1: DEFINING THE CLASSES

```
[Serializable]
public class Date
{
    private ushort year;
    private byte month, day;
    private DayOfWeek nameOfDay;

    public Date(int year, int month, int day)
    {
        this.year = (ushort)year;
        this.month = (byte)month;
        this.day = (byte)day;
        this.nameOfDay = (new DateTime(year, month, day)).DayOfWeek;
    }

    public Date(Date d)
    {
        this.year = d.year; this.month = d.month;
        this.day = d.day; this.nameOfDay = d.nameOfDay;
    }
```

STEP 1: DEFINING THE CLASSES

```
public int Year { get { return year; } }
public int Month { get { return month; } }
public int Day { get { return day; } }

// return this minus other, as of usual birthday calculations.
   public int YearDiff(Date other)
       {
           if (this.Equals(other))
               return 0;
           else if ((new Date(other.year, other.month,
  other.day)).IsBefore(other))
               return 0;
           else return this.year - other.year; ;

       }
```

STEP 1: DEFINING THE CLASSES

```
public override bool Equals(Object obj)
  {
     if (obj == null)
        return false;
     else if (this.GetType() != obj.GetType())
        return false;
     else if (ReferenceEquals(this, obj))
        return true;
     else if (this.year == ((Date)obj).year &&
              this.month == ((Date)obj).month &&
              this.day == ((Date)obj).day)
        return true;
     else return false;
  }
```

STEP 1: DEFINING THE CLASSES

```
public bool IsBefore(Date other)
    {
        return
          this.year < other.year ||
          this.year == other.year && this.month < other.month ||
          this.year == other.year && this.month == other.month && this.day < other.day;
    }
    public static DateToday
    {
        get
        {
            DateTime now = DateTime.Now;
            return new Date(now.Year, now.Month, now.Day);
        }
    }

    public override string ToString()
    {
        return string.Format("{0} {1}.{2}.{3}", nameOfDay, day, month, year);
    }
}
```

STEP 1: DEFINING THE CLASSES

```
[Serializable]
public class Person
{
    private string name;
    private int age;    // Redundant
    private Date dateOfBirth, dateOfDeath;
    public Person(string name, Date dateOfBirth)
    {
        this.name = name;
        this.dateOfBirth = dateOfBirth;
        this.dateOfDeath = null;
        age = Date.Today.YearDiff(dateOfBirth);
    }
    public Date DateOfBirth
    {
        get { return new Date(dateOfBirth); }
    }
    public int Age
    {
        get { return Alive ? age : dateOfDeath.YearDiff(dateOfBirth); }
    }
```

STEP 1: DEFINING THE CLASSES

```
public bool Alive
  {
     get { return dateOfDeath == null; }
  }

  public void Died(Date d)
  {
     dateOfDeath = d;
  }

  public void Update()
  {
     age = Date.Today.YearDiff(dateOfBirth);
  }

  public override string ToString()
  {
     return "Person: " + name +  "  *" + dateOfBirth +   (Alive ? "" : "  +" + dateOfDeath) +
            "  Age: " + Age;
  }
```

STEP 1: DEFINING THE CLASSES

```
[OnSerializing()]
   internal void OnSerializingMethod(StreamingContext context)
   {
      Console.WriteLine("serializing . . . .");
   }

   [OnSerialized()]
   internal void OnSerializedMethod(StreamingContext context)
   {
      Console.WriteLine("Done !");
   }
   [OnDeserializing()]
   internal void OnDeserializingMethod(StreamingContext context)
   {
      Console.WriteLine("deserializing . . . .");
   }
   [OnDeserialized()]
   internal void OnDeserializedMethod(StreamingContext context)
   {
      Console.WriteLine("obj Ready");
   }}
```

STEP 2: ATTEMPTING TO SERIALIZE AND DESERIALIZE

```
class Client
{
    public static void Main()
    {
        Person p = new Person("Peter", new Date(1936, 5, 11));
        p.Died(new Date(2007, 5, 10));
        Console.WriteLine("{0}", p);
        using (FileStream strm =     new FileStream("person.dat", FileMode.Create))
        {
            IFormatter fmt = new BinaryFormatter();
            fmt.Serialize(strm, p);
        }
        // ----------------------------------------------
        p = null;
        Console.WriteLine("Reseting person");
        // ----------------------------------------------
        using (FileStream strm =   new FileStream("person.dat", FileMode.Open))
        {
            IFormatter fmt = new BinaryFormatter();
            p = fmt.Deserialize(strm) as Person;
        }
        Console.WriteLine("{0}", p);    }}
```

OUTPUT

```
Person: Peter  *Monday 11.5.1936  +Thursday 10.5.2007  Age: 85
serializing . . . .
Done !
Resetting person
deserializing . . . .
obj Ready
Person: Peter  *Monday 11.5.1936  +Thursday 10.5.2007  Age: 85

Press any key to close this window . . .
```

THE “NONSERIALIZED” ATTRIBUTE

- Put [Serializable] on top of the class. For those attributes which you don't want to serialize put [NonSerialized] on them.

A CODE UPDATE TO SEE

```
[NonSerialized]
    private string name;
```

Guess the output ??

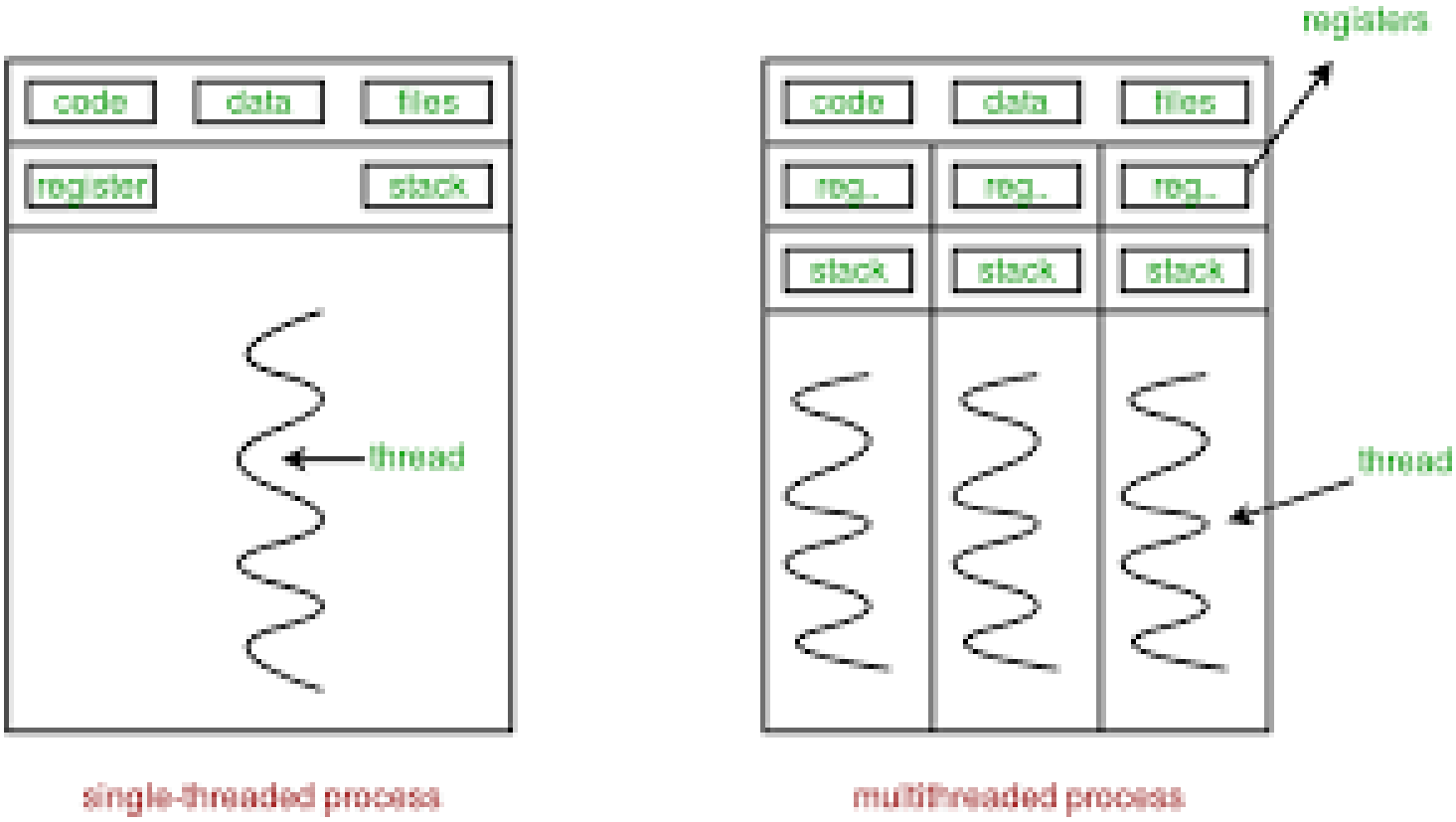
Person: Peter *Monday 11.5.1936 +Thursday 10.5.2007 Age: 71
serializing . . . .
Done !
Reseting person
deserializing . . . .
obj Ready
Person: *Monday 11.5.1936 +Thursday 10.5.2007 Age: 71

C:\Users\Dr Ali\source\repos\ConsoleApp1\ConsoleApp1\bin\Debug\netcoreapp3.1\ConsoleApp1.exe (process 10212) exited with code 0.
Press any key to close this window . . .

SERIALIZATION BENEFITS

- Serialization allows us to transfer objects through a network by converting it into a byte stream.
- It also helps in preserving the state of the object.
- Deserialization requires less time to create an object than an actual object created from a class. hence serialization saves time.
- One can easily clone the object by serializing it into byte streams and then deserializing it.
- Serialization helps to implement persistence in the program. It helps in storing the object directly in a database in the form of byte streams. This is useful as the data is easily retrievable whenever needed.
- To create a clone of the original object as a backup while working on the main object.
- A set of objects can easily be copied to the system's clipboard and then pasted into the same or another application

Threading


code
data
files
register
stack
thread
single-threaded process
registers
code
data
files
reg.
reg.
reg.
stack
stack
stack
thread
multithreaded process

WHAT IS A THREAD?

- A thread is nothing more than a process.
- On the computer, a thread is a process moving through time.
- The process performs sets of sequential steps, each step executing a line of code.
- Because the steps are sequential, each step takes a given amount of time.
- The time it takes to complete a series of steps is the sum of the time it takes to perform each programming step.

WHAT ARE MULTITHREADED APPLICATIONS (NETWORK APPLICATIONS)?

- For a long time, most programming applications were single-threaded.
- That means there was only one thread in the entire application.
- You could never do computation A until completing computation B.
- A multithreaded application allows you to run several threads, each thread running in its own process. So theoretically you can run step 1 in one thread and at the same time run step 2 in another thread.
- At the same time you could run step 3 in its own thread, and even step 4 in its own thread
- Theoretically, if all four steps took about the same time, you could finish your program in a quarter of the time it takes to run a single thread (assuming you had a 4 processor machine)
- An Unusual Analogy
  - Human Body

In case of single processor

CONCURRENCY !!

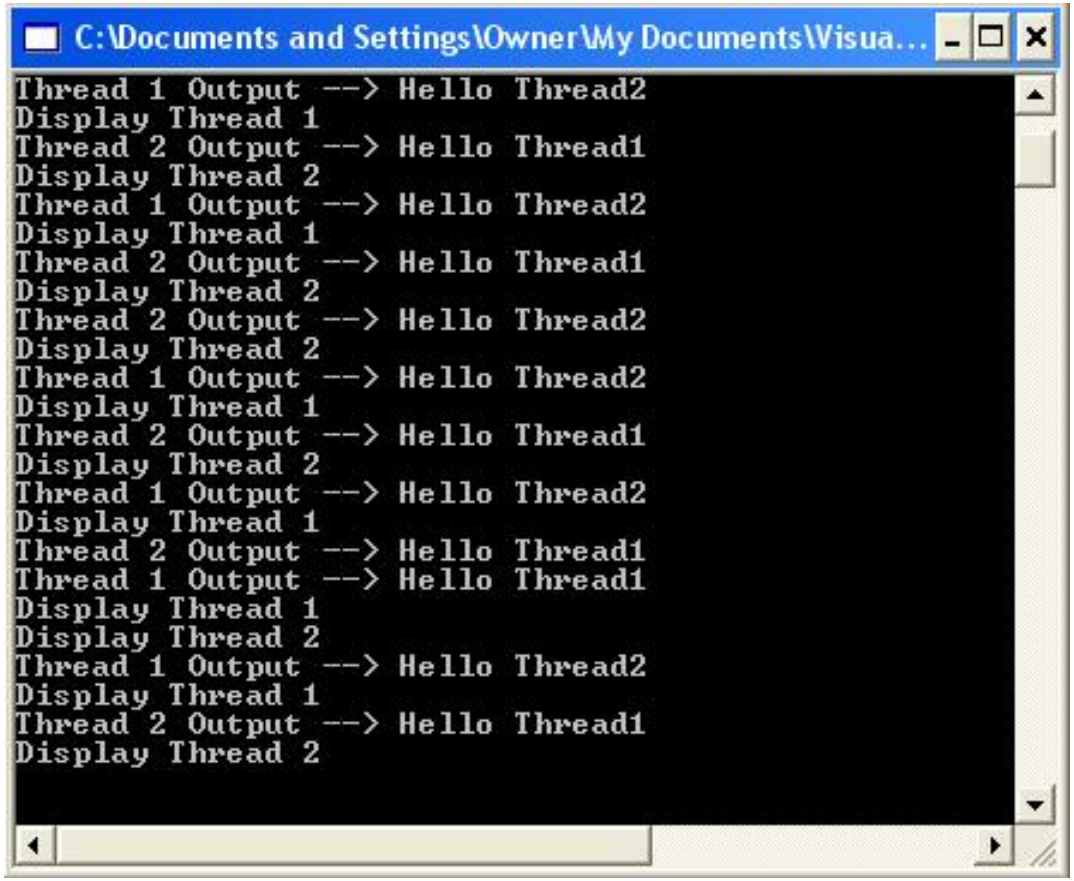

Can't wait,
lets Code !!

- We are aiming to Create two threads sharing a common variable in memory

```
using System;
using System.IO;
class MThread_APP
{
   // used to indicate which thread we are in
   private static string _threadOutput = "";
   private static bool _stopThreads = false;
   static void DisplayThread1()
   {
      while (_stopThreads == false)
      {
         Console.WriteLine("Display Thread 1");
         // Assign the shared memory to a message about thread #1
         _threadOutput = "Hello Thread1";
         Thread.Sleep(1000);  // simulate a lot of processing
         // tell the user what thread we are in thread #1, and display shared memory
         Console.WriteLine("Thread 1 Output --> {0}", _threadOutput);
      }   }
```

```
static void DisplayThread2()
  {
     while (_stopThreads == false)
     {
        Console.WriteLine("Display Thread 2");
    // Assign the shared memory to a message about thread #2
        _threadOutput = "Hello Thread2";
        Thread.Sleep(1000);  // simulate a lot of processing
        // tell the user we are in thread #2
        Console.WriteLine("Thread 2 Output --> {0}",
            _threadOutput);   }
  }
  public static void Main()
  {
     Thread thread1 = new Thread(new
           ThreadStart(DisplayThread1));
     Thread thread2 = new Thread(new              ThreadStart(DisplayThread2));
     // start them
     thread1.Start();
     thread2.Start();}}
```

OUTPUT
Any comment !!



EXPLANATION

- The code theoretically is executing the two methods DisplayThread1 and DisplayThread2, simultaneously.
- Each method shares the variable, _threadOutput.
- Race condition !
  - Each method shares the variable, _threadOutput. So it is possible that _threadOutput is assigned a value "Hello Thread1" in thread #1 and displays _threadOutput
  - Somewhere in between the time thread #1 assigns it and displays it, thread #2 assigns _threadOutput the value "Hello Thread2"
  - Thread#1 assigns and thread#2 displays !!
  - How to organize that !!
  - Use **Lock**

THREAD SAFE CODE

- The best way to avoid race conditions is to write thread-safe code
- Make two instances (don't share)
- Let any one to win the race!
- The way we prevent one thread from affecting the memory of the is called *locking.*
- So any thread trying to modify a field of the class while inside the lock will be *blocked*
- Blocking means that the thread trying to change the variable will sit and wait until the lock is released on the locked thread.
- The thread is released from the lock upon reaching the last bracket in the lock { } construct.

```
using System;
using System.IO;
class MThread_APP
{
   // used to indicate which thread we are in
   private  string _threadOutput = "";
   private  bool _stopThreads = false;
   public MThread_APP()   {
      Thread thread1 = new Thread(new ThreadStart(DisplayThread1));
      Thread thread2 = new Thread(new ThreadStart(DisplayThread2));
      // start them
      thread1.Start();
      thread2.Start();
   }
    void DisplayThread1()
   {
      while (_stopThreads == false)   {
         // lock on the current instance of the class for thread #1
         lock (this)     {
            Console.WriteLine("Display Thread 1");
            _threadOutput = "Hello Thread1";
            Thread.Sleep(1000);  // simulate a lot of processing
// tell the user what thread we are in thread #1
            Console.WriteLine("Thread 1 Output --> {0}", _threadOutput);          }// lock released  for thread #1
here
      }   }
```

```
void DisplayThread2()   {
      while (_stopThreads == false)     {

// lock on the current instance of the class for thread #2
         lock (this)        {
            Console.WriteLine("Display Thread 2");
            _threadOutput = "Hello Thread2";
            Thread.Sleep(1000);  // simulate a lot of processing
                              // tell the user what thread we are in thread #1
Console.WriteLine("Thread 2 Output --> {0}", _threadOutput);
         } // lock released  for thread #2 here
      }   }}
class MainClass
{
   public static void Main()
   {
     new MThread_APP();
   }}
```

OUTPUT

C:\Users\Dr. Ali\source\repos\binarystreams\binarystreams\bin\D

```
Display Thread 2
Thread 2 Output --> Hello Thread2
Display Thread 1
Thread 1 Output --> Hello Thread1
Display Thread 2
Thread 2 Output --> Hello Thread2
Display Thread 1
Thread 1 Output --> Hello Thread1
Display Thread 2
Thread 2 Output --> Hello Thread2
Display Thread 1
Thread 1 Output --> Hello Thread1
Display Thread 2
Thread 2 Output --> Hello Thread2
Display Thread 1
Thread 1 Output --> Hello Thread1
Display Thread 2
Thread 2 Output --> Hello Thread2
Display Thread 1
```

LETS HAVE AN EXAMPLE

WHAT CAN WE USE TO AVOID RACE CONDITION?

- C# provides great facilities which can be used to manage shared areas
- The `AutoResetEvent` class
- Two main methods
  - `WaitOne()` → for the associated obj, wait here don't proceed executing !
  - `Set()` → for the associated obj, ok you can proceed executing

```
using System;
using System.Threading;
class MThread_APP
{
    // used to indicate which thread we are in
    private string _threadOutput = "";
    private bool _stopThreads = false;
    public MThread_APP()
    {
        Thread thread1 = new Thread(new
ThreadStart(DisplayThread_1));
        Thread thread2 = new Thread(new
ThreadStart(DisplayThread_2));
        // start them
        thread1.Start();
        thread2.Start();
    }
    AutoResetEvent _blockThread1 = new AutoResetEvent(false);
    AutoResetEvent _blockThread2 = new AutoResetEvent(true);
```

```
 void DisplayThread_1() {
        while (_stopThreads == false) {
            // block thread 1  while the thread 2 is executing
            _blockThread1.WaitOne();
            // Set was called to free the block on thread 1,
continue executing the code
            Console.WriteLine("Display Thread 1");
            _threadOutput = "Hello Thread 1";
            Thread.Sleep(1000);  // simulate a lot of processing

            // tell the user what thread we are in thread #1
            Console.WriteLine("Thread 1 Output --> {0}",
_threadOutput);

            // finished executing the code in thread 1, so unblock
thread 2
            _blockThread2.Set();
        }
    }
```

```
 void DisplayThread_2()
    {
        while (_stopThreads == false)
        {
            // block thread 2  while thread 1 is executing
            _blockThread2.WaitOne();

            // Set was called to free the block on thread 2,
continue executing the code
            Console.WriteLine("Display Thread 2");
            _threadOutput = "Hello Thread 2";
            Thread.Sleep(1000);  // simulate a lot of processing

            // tell the user we are in thread #2
            Console.WriteLine("Thread 2 Output --> {0}",
_threadOutput);

            // finished executing the code in thread 2, so
unblock thread 1
            _blockThread1.Set();
        }    }    }
```

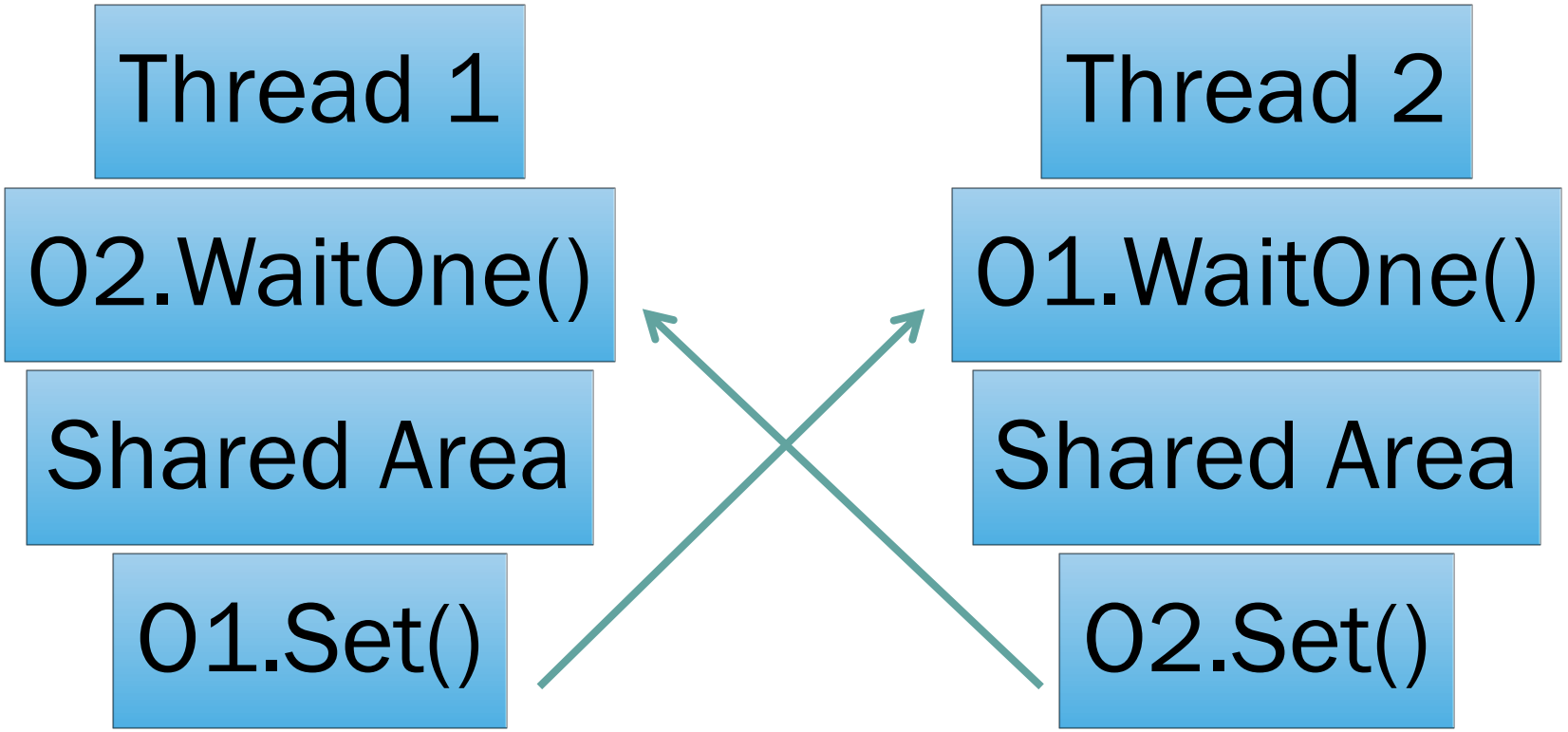
OUTPUT, WHY THREAD 2 BEGINS?!

C:\Users\Dr Ali\source\repos\ConsoleApp1\ConsoleApp1\bin\Debug\netcoreapp3.1\ConsoleApp1.exe

```
Display Thread 2
Thread 2 Output --> Hello Thread 2
Display Thread 1
Thread 1 Output --> Hello Thread 1
Display Thread 2
Thread 2 Output --> Hello Thread 2
Display Thread 1
Thread 1 Output --> Hello Thread 1
Display Thread 2
Thread 2 Output --> Hello Thread 2
Display Thread 1
Thread 1 Output --> Hello Thread 1
Display Thread 2
Thread 2 Output --> Hello Thread 2
Display Thread 1
```

```
class MainClass
{
    public static void Main()
    {
        new MThread_APP();
    }
}
```

O2.Set Lets O2 to Proceed the Code After O2.WaitOne And Vice Versa



PERFORMANCE USING THREADS

- Keep in mind, creating more threads is not related to processing speed or performance.
- All threads share the same processor and resources a machine has.
- In cases of multiple threads, the thread scheduler with the help of the operating system schedules threads and allocates a time for each thread.
- But in a single processor machine, only one thread can execute at a time.
- The rest of the threads have to wait until the processor becomes available.
- Creating more than a few threads on a single processor machine may create a resource bottleneck if not managed properly.
- **So don't "Thread" Everything**

THANK U