Generic Delegate.

File IO and Serialization —

→ Directory class :— Create

→ File → WriteAll() , ReadAll()

System.IO

System.IO

class Stream :- abstract.

- class FileStream

  ↓ we can persist the data on m/c / hard drive.

- class NetworkStream

  → data can be transferred over the network

- class CryptoStream

  → while persisting data on HDD OR transferring data via network using this class we can Encrypt and Decrypt the data.

CLR → [Serializable] class : Attribute

↓

it gives permission to CLR to persist the class object in txt file / on the HDD.

_No, _Name _Address → field persist

↳ [NonSerializable]

→ Denines the permission to CLR

```
public   enum   Days
{
        Mon , Tue, Wed, Thu, Fri

 };

public void  AvailableDays((Days.Mon |   -> pipe operator
                            Days.Wed |
                            Days.Thu));

signature
public void AvailableDays(Days day)
{

}
```

Single Parameter.

Collection $\rightarrow$

```
<? xml ? >

✓ <Emp>
      <Id> 101 </Id>
      <Name> John Conner </Name>
      <Addr> Earth </Addr>.
</Emp>
```

→ XML Schema

CLR

```
<Emp>
      <Id>
```

Reflection is a technique where you can read Type MetaData at Runtime / dynamically and you can invoke the assemblies / modules / functionalities at RunTime. / dynamically.

```
namespace Demo
public
class CMath{

public int Add
   (int x , int y)
{
   return x+y;
}

public int sub
   (int x, int y)
{
   return x-y;
}
}
```

DemoCMath.dll

```
Assembly asm = Assembly.LoadFrom ("Path CMath.dll")
Type[] allTypes = asm.GetTypes()
   class, enum, struct, delegate
   interface, abstract, static
      Type type = allTypes[i]
                    CMath
   = Type. FullName → "ooDemo. CMath"
   = type. Name → CMath
     type. isPulic = true
     type. is Private = false
     type. Is Static = false
     type. is Abstract = false.
     type. is Sealed = false.
           . is Inherited = false.
Method Info [] allmethods = type. GetMethods()
   MethodInfo meth = allMethods[i]
                               Add
   meth. Name ————→
ParameterInfo [] paras = meth.GetParameter()
   ParameterInfo para = paras[i]
                 int x , int y
```