Generic Delegate.

File IO and Serialization —

→ Directory class :— Create

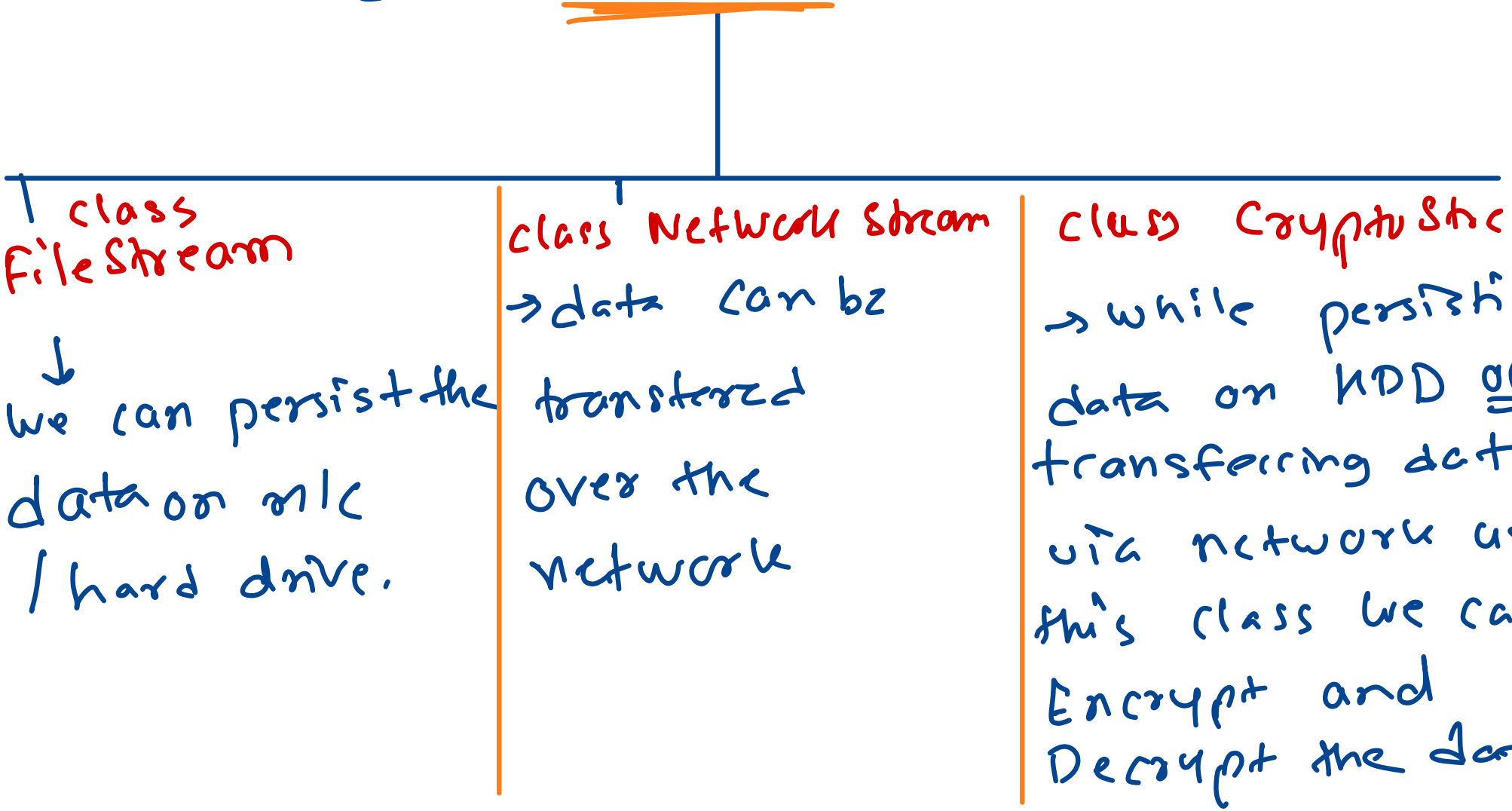
→ File → WriteAll(), ReadAll()

↓

System.IO

System.IO

class Stream :- abstract.



CLR → [Serializable] class : Attribute

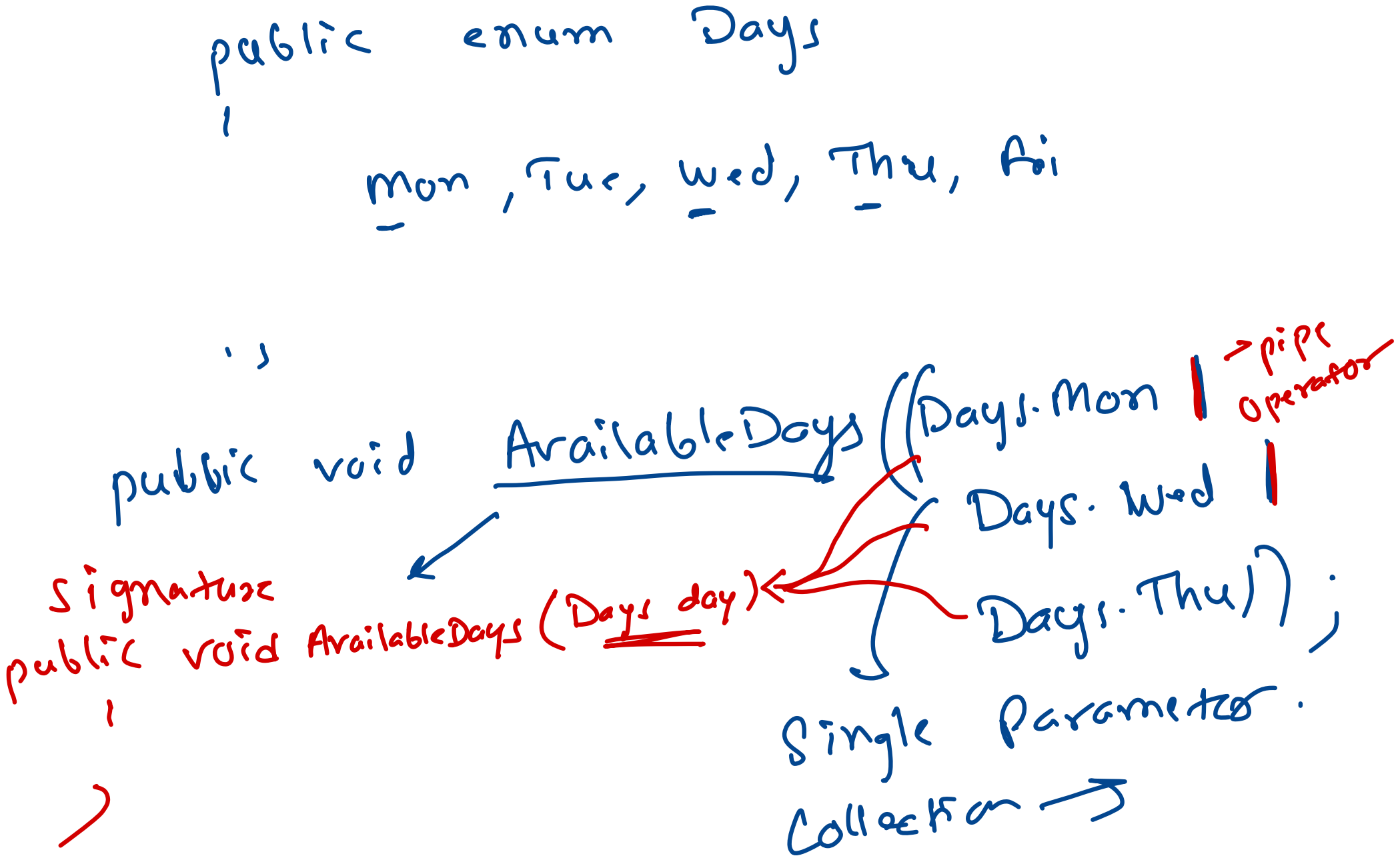
↓

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

_No, _Name, _Address → field persist

↳ [Non Serializable]

→ Denies the permission to CLR

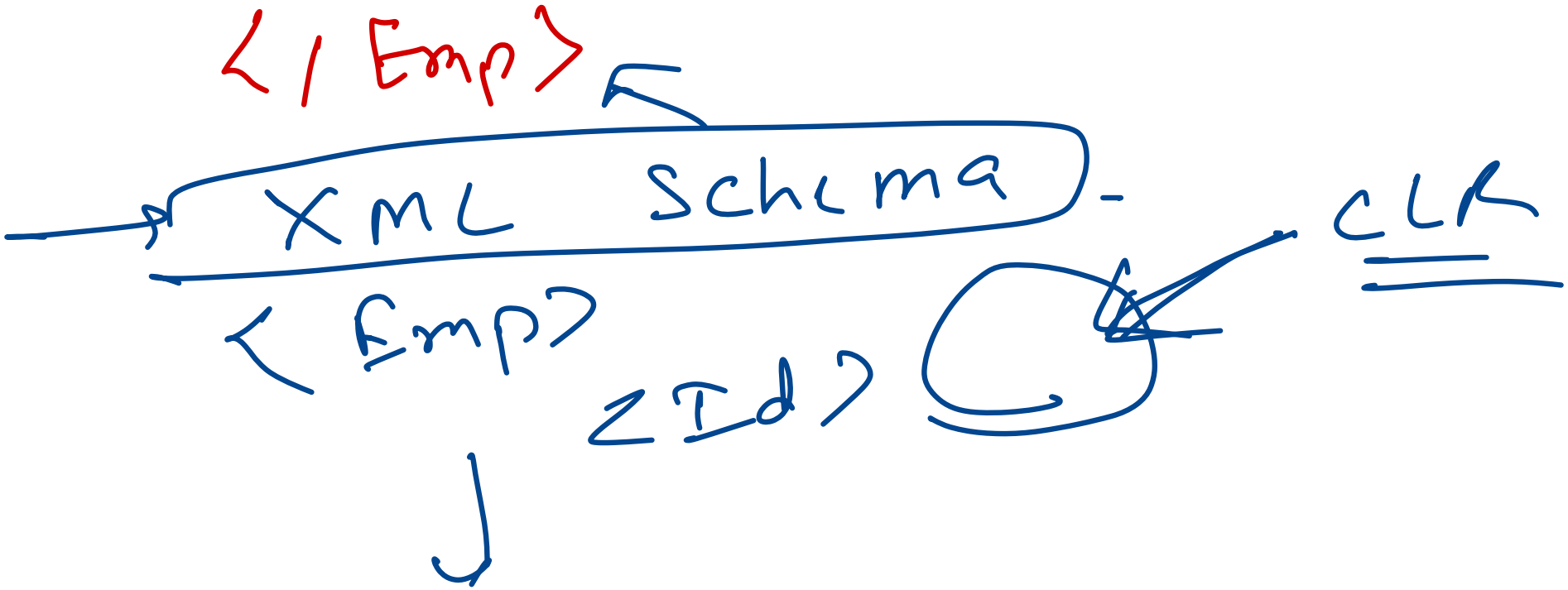

public enum Days
{
Mon, Tue, Wed, Thu, Fri
}
pubbic void AvailableDays((Days.Mon | Days.Wed | Days.Thu));
pipe operator
Signature
public void AvailableDays(Days day)
{
}
Single Parameter.
Collection

<?xml ?>

<Emp>

<Id> 101 </Id>
<Name> John Conner </Name>
<Addr> Earth </Addr>.

</Emp>



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 → "DoDemo.Cmath"
= type.Name → CMath
type.isPulic = true
type.isPrivate = false
type.IsStatic = false
type.IsAbstract = false.
    .isSealed = false.
    .isInherited = false.
```

```
MethodInfo[] allmethods = type.GetMethods()
MethodInfo meth = allmethods[i]
```

Add

```
meth.Name → Add
ParameterInfo[] paras = meth.GetParameter()
ParameterInfo para = para[i]
```

int x, int y