Day 4 : .NET & C# Basics

Reflection

- All .NET assemblies have metadata information stored about the types defined in modules
- Allows a program to modify itself or inspect during the runtime
- Namespace is System.Reflection
- Allows examining various types in assembly and instantiate these types
- Allows late binding to properties and methods

System.Type class

- Base class for all reflection class and properties
- Primary way to access metadata

```
// Using GetType to obtain type information:
int i = 42;
System.Type type = i.GetType();
System.Console.WriteLine(type);

//Output
System.Int32
```

```
static void Main(string[] args)
{
    // Using GetType to obtain type information:
    int i = 42;
    System.Type type = i.GetType();
    System.Console.WriteLine(type);

}
```

type {Name = "Int32" FullName = "System.Int32"}

Assembly	{System.Private.CoreLib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=7cec85d7bea7798e}
AssemblyQualifiedName	"System.Int32, System.Private.CoreLib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=7cec85d7bea7798e"
Attributes	Public | SequentialLayout | Sealed | Serializable | BeforeFieldInit
BaseType	{Name = "ValueType" FullName = "System.ValueType"}
ContainsGenericParameters	false
CustomAttributes	Count = 3
DeclaredConstructors	{System.Reflection.ConstructorInfo[0]}
DeclaredEvents	{System.Reflection.EventInfo[0]}
DeclaredFields	{System.Reflection.FieldInfo[3]}
DeclaredMembers	{System.Reflection.MemberInfo[38]}
DeclaredMethods	{System.Reflection.MethodInfo[35]}
DeclaredNestedTypes	{System.Reflection.TypeInfo.<get_DeclaredNestedTypes>d__22}
DeclaredProperties	{System.Reflection.PropertyInfo[0]}
DeclaringMethod	'((System.RuntimeType)type).DeclaringMethod' threw an exception of type 'System.InvalidOperationException'
DeclaringType	null

Assembly class

- Can be used to define and load assemblies, locate a type from the assembly and create an instance of it

```
// Using GetType to obtain type information:
var assembly = System.Reflection.Assembly.GetAssembly(Type type);

//loading an assembly by name
var assembly = System.Reflection.Assembly.Load("System.String");
```

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        var assembly = Assembly.GetExecutingAssembly();
    }

}
```

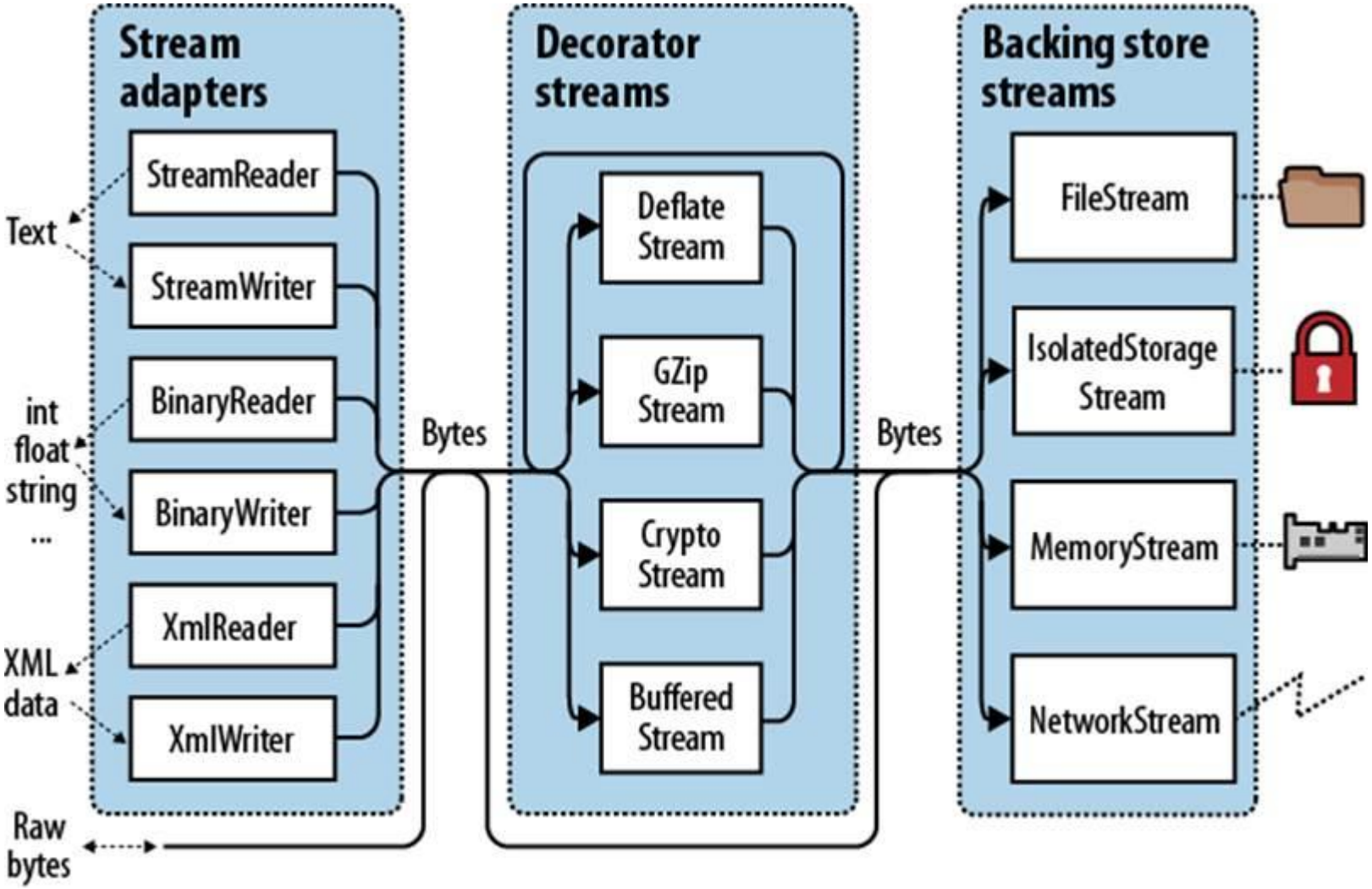
assembly {Basics, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null}

CodeBase	"file:///E:/Training/Curriculam/CSharp/Day1/Work/Basics/bin/Debug/netcoreapp3.1/Basics.dll"
CustomAttributes	Count = 10
DefinedTypes	{System.RuntimeType[1]}
EntryPoint	{Void Main(System.String[])}
EscapedCodeBase	"file:///E:/Training/Curriculam/CSharp/Day1/Work/Basics/bin/Debug/netcoreapp3.1/Basics.dll"
ExportedTypes	{System.Type[0]}
FullName	"Basics, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"
GlobalAssemblyCache	false
HostContext	0
ImageRuntimeVersion	"v4.0.30319"
IsCollectible	false
IsDynamic	false
IsFullyTrusted	true
Location	"E:\\Training\\Curriculam\\CSharp\\Day1\\Work\\Basics\\bin\\Debug\\netcoreapp3.1\\Basics.dll"
ManifestModule	{Basics.dll}

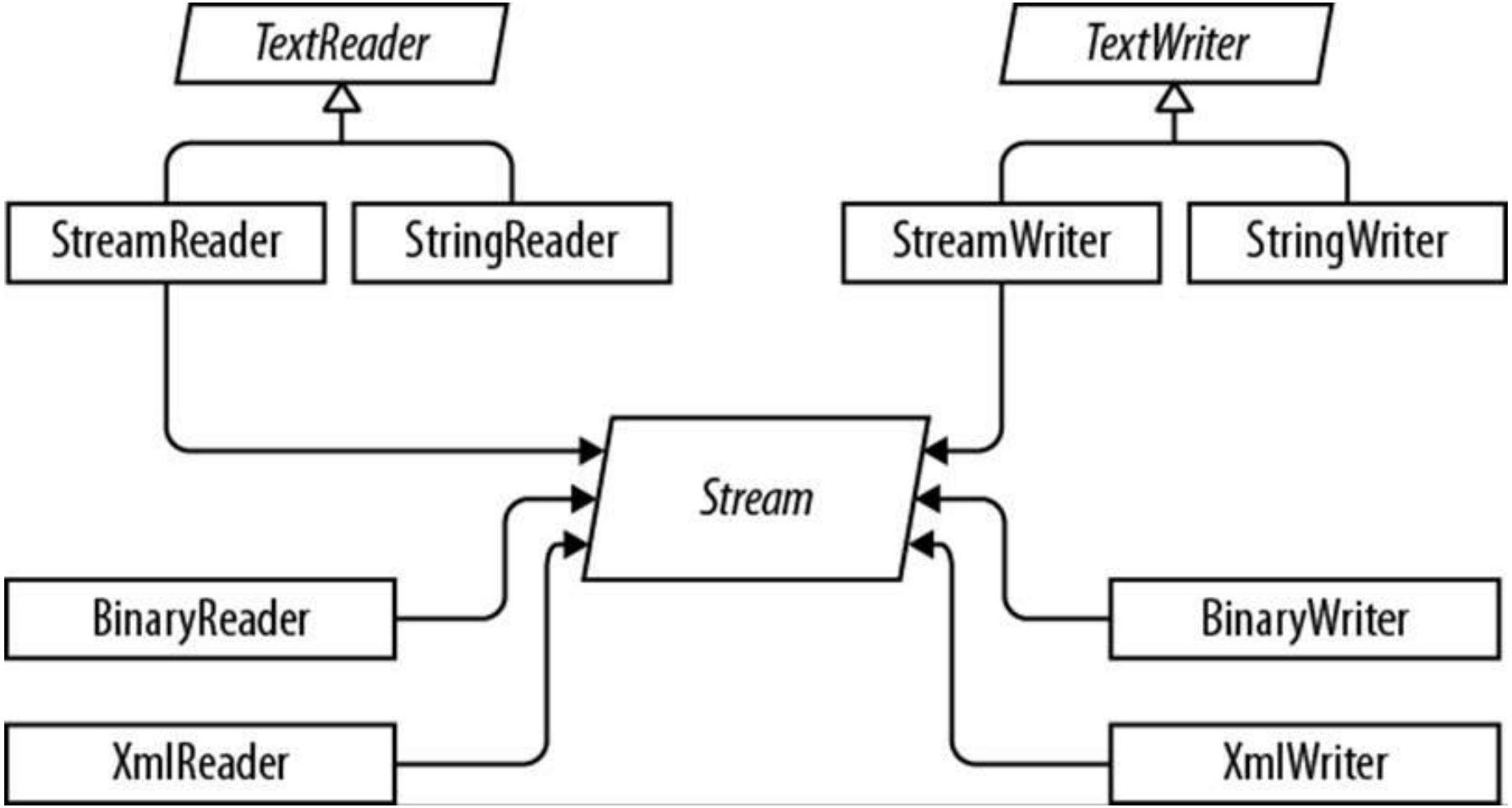
Invoking a method

```
// dynamically load assembly from file Test.dll
Assembly testAssembly = Assembly.LoadFile(@"c:\Test.dll");

// get type of class from just loaded assembly
Type objClass1 = testAssembly.GetType("Test.Class1");

// create instance of the class
object classInstance = Activator.CreateInstance(objClass1);
```

I/O Architecture

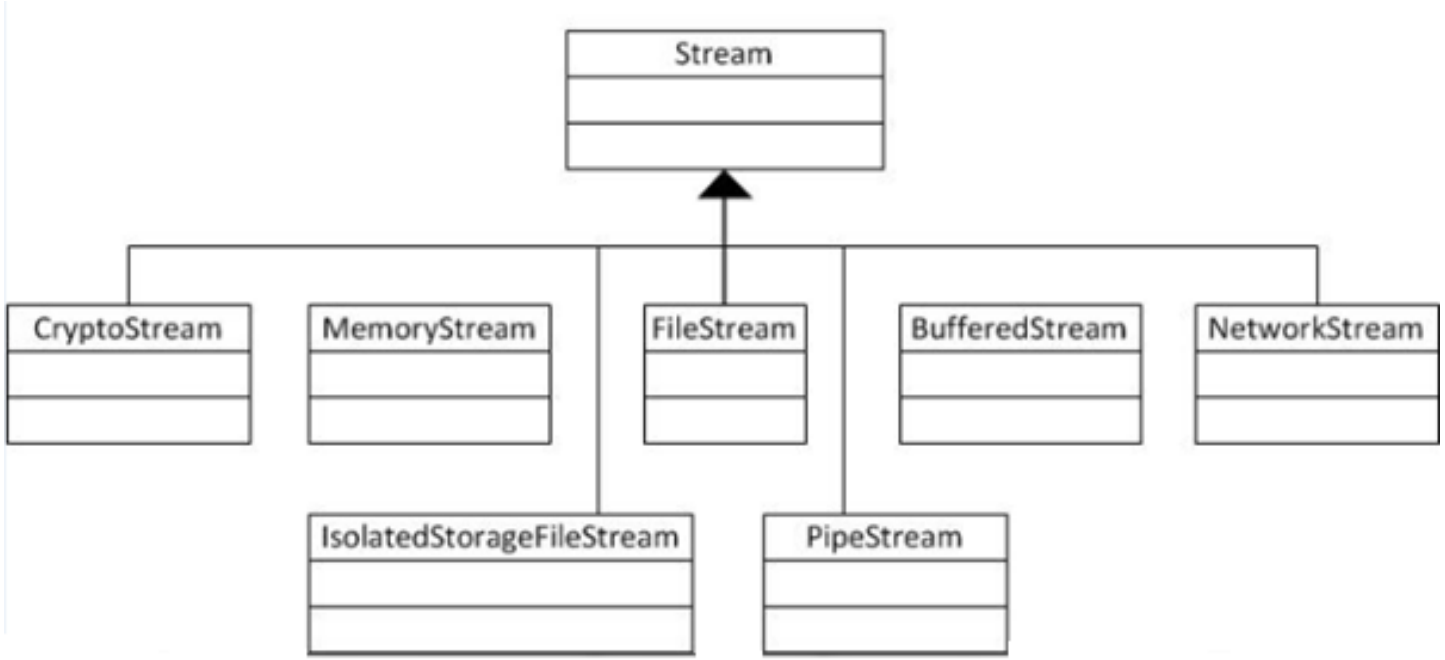


Stream Readers & Writers



I/O Streams

- Stream
- FileStream
- MemoryStream
- BufferedStream
- NetworkStream
- PipeStream
- CryptoStream



Stream Readers & Writers

- StreamReader
- StreamWriter
- BinaryReader
- BinaryWriter

Example

```
// using a StremWriter object to write data to a file
// use this when you have string or char data,
      using (StreamWriter writer = new StreamWriter(“test.txt”))
      {
         writer.Write(“Sample text");
      }

//using FileStream for writing
        FileStream fWriter = new FileStream(“test.txt”, FileMode.Create);
        // Store the text in a byte array with UTF8 encoding
        byte[] arr = Encoding.UTF8.GetBytes(text);

        // Using the Write method write the encoded byte array to the textfile
        fWriter.Write(arr, 0, text.Length);
```

Choosing a FileMode