컴퓨터정보과
C# 프로그래밍

12주차 파일처리

강의 순서

1. C# 환경설치 / C# 기본 구조
2. 클래스 기본(필드) + 변수,자료형
3. 클래스 기본(메소드) + 연산자,수식
4. 클래스 기본(메소드) + 제어문
5. 배열/리스트/딕셔너리
6. 클래스 기본: 접근제한자 (한정자) + 프로퍼티(속성)
7. 클래스 심화:상속
8. 클래스 심화:인터페이스/추상 클래스
9. 예외처리/일반화/…
10. **파일처리**
11. UI (Winform or WPF)
12. LINQ/Delegate/Lamda/…

복습 및 11주차 추가요소

- 예외와 예외처리
  - 예외 : 실행시간 중에 발생한 오류
    - 모든 예외는 Exception 클래스를 상속 받는다.
  - 예외처리
    - try – catch – finally
  - 예외 발생
    - throw new 예외클래스_생성자()

- 일반화 프로그래밍
  - Type Parameter를 이용해 클래스나 메소드 내의 동작을 특정 데이터 타입을 정하지 호출이나 생성시 데이터 타입을 지정할 수 있는 방식
  - Type Parameter는 1개 이상 지정할 수 있음.
  - 실제로 원하는 동작을 하기 위해서는 제약이나 여러 기법들이 필요함.

- ref, out
  - 변수의 참조를 전달하기 위한 수단.

- 클래스 만들기 연습

파일 & 폴더

프로그래밍이 생성한 데이터를 보관하기

파일 & 폴더

- File : 컴퓨터 저장장치(HDD, SSD, USB, …)에 기록하는 데이터 단위
- Folder (Directory) : File을 관리하는 단위

클래스	설명	종류
File	파일의 생성, 복사, 삭제, 이동, 조회	정적 메소드
FileInfo		인스턴스 메소드
Directory	폴더의 생성, ~~복사~~, 삭제, 이동, 조회	정적 메소드
DirectoryInfo		인스턴스 메소드

파일 & 폴더

class / 기능	File	FileInfo	Directory	DirectoryInfo
생성	Create()	Create()	CreateDirectory()	Create ()
복사	Copy()	CopyTo()	-	-
삭제	Delete()	Delete()	Delete()	Delete()
이동	Move()	MoveTo()	Move()	MoveTo()
존재여부	Exists()	Exists	Exists()	Exists
속성조회	GetAttributes()	GetAttributes	GetAttributes()	GetAttributes
하위 폴더 조회	-	-	GetDirectories()	GetDirectories()
하위 파일 조회	-	-	GetFiles()	GetFiles()

파일

class / 기능	File	FileInfo
생성	`FileStream fs = File.Create("s.dat");`	`FileInfo f = new FileInfo("s.data");` `FileStream fs = f.Create();`
복사	`File.Copy("s.dat", "t.dat");`	`FileInfo s = new FileInfo("s.data");` `FileInfo t = s.CopyTo("t.dat");`
삭제	`File.Delete("s.dat");`	`FileInfo s = new FileInfo("s.data");` `s.Delete();`
이동	`File.Move("s.dat“ , "t.dat");`	`FileInfo s = new FileInfo("s.data");` `File.MoveTo("t.dat");`
존재여부	`if(File.Exist("s.dat")) { … }`	`FileInfo s = new FileInfo("s.data");` `if(s.Exist) { … }`
속성조회	`Console.WriteLine(` `  File.GetAttributes("s.dat"));`	`FileInfo s = new FileInfo("s.data");` `Console.WriteLine(s.GetAttributes);`

폴더 (디렉토리)

기능 \ class	Directory	DirectoryInfo
생성	DirectoryInfo dir = Directroy. CreateDirectory("s");	DirectoryInfo dir = new DirectoryInfo("s"); dir.Create()
삭제	Directroy.Delete("s");	DirectoryInfo dir = new DirectoryInfo("s"); dir.Delete()
이동	Directroy.Move("s " , "t");	DirectoryInfo dir = new DirectoryInfo("s"); dir.MoveTo("t");
존재여부	if(Directroy.Exist("s")) { … }	DirectoryInfo dir = new DirectoryInfo("s"); if(dir.Exist) { … }
속성조회	Console.WriteLine(Directroy.GetAttributes("s"));	DirectoryInfo dir = new DirectoryInfo("s"); Console.WriteLine(dir.GetAttributes);
하위 폴더 조회	string[] dirs = Directory.GetDirectories("s"));	DirectoryInfo dir = new DirectoryInfo("s"); DirectoryInfo[] dirs = dir.GetDirectories());
하위 파일 조회	string[] files = Directory.GetFiles("s"));	DirectoryInfo dir = new DirectoryInfo("s"); FileInfo[] files = dir.GetFiles());

Stream

데이터가 흐르는 길

Stream 스트림

- 영어로 시내, 강, 또는 도로의 차선을 뜻함
- 파일을 다룰 때의 스트림 : 바이트 단위로 데이터가 흐르는 통로
- 메모리에서 하드 디스크로 데이터를 옮길 때, 스트림을 만들어 둘 사이를 연결한 뒤에 메모리의 데이터를 바이트 단위로 하드 디스크로 옮김 (반대도 마찬가지)
- System.IO.Stream 클래스
  - 입력/출력 스트림 역할 수행
  - 순차/임의 접근 방식 모두 지원
  - 추상 클래스임으로, 파생 클래스 이용해야 함
    - FileStream
    - NetworkStream
    - GZipStream
    - MemoryStream
    - BufferedStream

FileStream

```
 double value = 1234.56789069;
 byte[] wbytes = BitConverter.GetBytes(value);
 Stream outstream = new FileStream("t.dat", FileMode.Create);
 outstream.Write(wbytes, 0, wbytes.Length);
 outstream.Close();

byte[] rbytes = new byte[8];
Stream instream = new FileStream("t.dat", FileMode.Open);
instream.Read(rbytes, 0, rbytes.Length);
instream.Close();

double cnvvalue = BitConverter.ToDouble(rbytes, 0);
Console.WriteLine(cnvvalue);
```

Helper class

Helper Class

- FileStream: 반드시byte, byte[] 형식으로입출력해야함.
  - 데이터 입출력이 불편함
- 해당 타입에 맞도록 알아서 byte[] 배열에 저장하도록 또는 byte[] 배열에서 해당 타입에 맞게 읽어오도록 도와주는 클래스

- BinaryWriter / BinaryReader
  - 모든 데이터를 이진 데이터로 기록하고 읽기
  - 특정 프로그램들에서만 정확하게 읽고 쓸 수 있음

- StreamWriter / StreamReader
  - 모든 데이터를 문자 데이터로 기록하고 읽기
  - 모든 프로그램에서 읽고 쓸 수 있음

BinaryWriter/BinaryReaer

```
BinaryWriter bw = new BinaryWriter(new FileStream("b.dat", FileMode.Create));
bw.Write(1234);
bw.Write(123.4);
bw.Write("인하");
bw.Close();

BinaryReader br = new BinaryReader(new FileStream("b.dat", FileMode.Open));
int a = br.ReadInt32();
double b = br.ReadDouble();
string c = br.ReadString();
br.Close();

Console.WriteLine($"{a}, {b}, {c}");
```

StreamWriter/StreamReader

```
StreamWriter sw = new StreamWriter(new FileStream("s.txt", FileMode.Create));
sw.WriteLine(1234);
sw.WriteLine(123.4);
sw.WriteLine("인하");
sw.Close();

StreamReader sr = new StreamReader(new FileStream("s.txt", FileMode.Open));

while (false == sr.EndOfStream) {
    Console.WriteLine(sr.ReadLine());
}
sr.Close();
```

Serialization

- 클래스나 구조체의 복잡한 데이터를 파일에 읽고 쓰려면 어떻게 해야하는가.
  - 각 데이터를 하나씩 분해해서 저장하거나..
    - 기존 저장 방식
  - 한꺼번에 저장하거나...
    - 직렬화

- Serialization(직렬화)
  - 객체의 상태를 메모리나 저장장치에 저장이 가능한 0과 1의 순서로 바꾸는 것을 뜻함. (바이트 배열)

Serialization

- 직렬화 가능한 class 생성

```
[Serializable]
class Student
{
    public string Number;
    public string Name;

    public override string ToString()
    {
        return $"{Number}:{Name}";
    }
}
```

- 데이터 생성

```
var student = new Student() { Number = "1", Name = "김미영" };
```

Serialization

- 직렬화 수행 후 파일에 저장

```
Stream ws = new FileStream("sr.dat", FileMode.Create);
BinaryFormatter serializer = new BinaryFormatter();
serializer.Serialize(ws, student);
ws.Close();
```

- 파일에서 역직렬화 수행 후 복구

```
Stream rs = new FileStream("sr.dat", FileMode.Open);
BinaryFormatter deserializer = new BinaryFormatter();
var cnvstudent = (Student)deserializer.Deserialize(rs);
ws.Close();
```

기타

- using
- NameCard 예제
- Student 클래스의 데이터를 List나 Array인 경우 어떻게 파일에 저장할까?