컴퓨터정보과
C# 프로그래밍

10주차 다형성과 클래스 심화(인터페이스 & 추상클래스)

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 9주차 추가요소

- 상속
  - 코드의 재활용
  - 기존 코드의 확장
  - 기본 클래스(base class) & 파생 클래스(derived class)
    - 생성자/종료자의 실행 순서
  - Object
    - 최상위 클래스
  - base
  - 연산자
    - () 연산자
    - is 연산자
    - as 연산자
  - 접근제한자
    - protected

다형성 (polymorphism)

코드의 재사용과 확장

다형성, Polymorphism

- 다형성이란 프로그램 언어 각 요소들(상수, 변수, 식, 객체, 메소드 등)이 다양한 자료형(type)에 속하는 것이 허가되는 성질을 가리킨다.
  - 런타임에 파생 클래스의 객체가 메서드 매개 변수 및 컬렉션 또는 배열과 같은 위치에서 기본 클래스의 객체로 처리될 수 있습니다. 이러한 다형성이 발생하면 객체의 선언된 형식이 더 이상 해당 런타임 형식과 같지 않습니다.
  - 기본 클래스는 가상메서드를 정의 및 구현할 수 있으며, 파생 클래스는 이러한 가상 메서드를 재정의할 수 있습니다. 즉, 파생 클래스는 고유한 정의 및 구현을 제공합니다. 런타임에 클라이언트 코드에서 메서드를 호출하면 CLR은 객체의 런타임 형식을 조회하고 가상 메서드의 해당 재정의를 호출합니다. 소스 코드에서 기본 클래스에 대해 메서드를 호출하여 메서드의 파생 클래스 버전이 실행되도록 할 수 있습니다.

다형성 - C# | Microsoft Learn

변수 하이딩 (숨기기)

```
class Super
{
    public int Number = 100;
}

class Sub : Super
{
    public double Number = 200.0;

    public void Print()
    {
        Console.WriteLine(base.Number);
        Console.WriteLine(this.Number);
    }
}

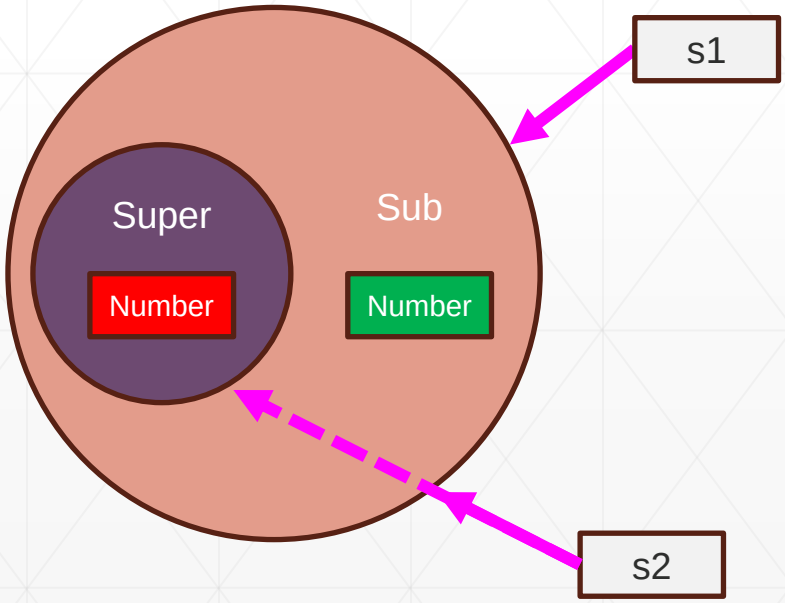
class Program
{
    static void Main(string[] args)
    {
        Sub s1 = new Sub();
        Console.WriteLine(s1.Number);

        Super s2 = (Super)s1;
        Console.WriteLine(s2.Number);

        s1.Print();
    }
}
```

new

```
C:₩WINDOWS₩system
200
100
100
200
계속하려면 아무 키나 누르십시오 . . .
```



메소드 오버라이딩(재정의)과 하이딩(숨기기)

- 가상 메소드는 디자이너에게 파생 클래스의 동작에 대한 다양한 선택권을 제공
  1. 파생 클래스는 재정의하지 않고 가장 가까운 기본 클래스 메서드를 상속할 수 있어 기존 동작을 유지하지만 추가 파생 클래스가 메서드를 재정의할 수 있습니다.
  2. 파생 클래스가 기본 클래스의 가상 메소드를 재정의하여 새로운 동작을 정의할 수 있습니다. (method overriding)
  3. 파생 클래스가 기본 클래스 구현을 숨기는 멤버의 새로운 비가상 구현을 정의할 수 있습니다. (method hiding)

- 파생(derived) 클래스는 기본(base) 클래스 멤버가 **virtual** 또는 **abstract**로 선언된 경우에만 기본 클래스 멤버를 재정의(overrideing)할 수 있습니다. 파생 멤버는 **override** 키워드를 사용하여 메서드가 가상 호출에 참여하도록 되어 있음을 명시적으로 나타내야 합니다.
- 파생(derived) 클래스가 기본(base) 클래스의 멤버와 동일한 이름을 갖는 멤버를 갖도록 하려면 **new** 키워드를 사용하여 기본 클래스 멤버를 숨길 수(hiding) 있습니다.
- 재정의한 파생 클래스는 계속해서 **base** 키워드를 사용하여 기본 클래스에 대한 메서드 또는 속성에 액세스할 수 있습니다. 다음 코드는 예제를 제공합니다.

다형성 - C# | Microsoft Learn

메소드 오버라이딩과 하이딩 (계속)

```
class Pet {
    public void Sleep() //일반 메소드
    {
        Console.WriteLine("Zzzz");
    }

    public virtual void Eat() //가상 메소드
    {
        Console.WriteLine("우걱우걱");
    }
}

class Cat : Pet {
    public new void Sleep() //하이딩 가능
    {
        Console.WriteLine("고양이가 잔다 골골");
    }

    public override void Eat() //오버라이딩 가능
    {
        Console.WriteLine("고양이가 먹는 뇸뇸");
    }
}

class Dog : Pet {
    public new override void Sleep() //하이딩만 가능
    {
        Console.WriteLine("멍멍이가 잔다 드르렁");
    }

    public new void Eat() //하이딩도 가능
    {
        Console.WriteLine("멍멍이가 먹는 냠냠");
    }
}
```

```
internal class Program
{
    static void Main(string[] args)
    {
        Pet cat = new Cat();
        Pet dog = new Dog();

        Console.WriteLine("------------------");
        cat.Sleep();
        cat.Eat();
        Console.WriteLine("------------------");
        dog.Sleep();
        dog.Eat();
        Console.WriteLine("------------------");
        ((Cat)cat).Sleep();
        ((Cat)cat).Eat();
        Console.WriteLine("------------------");
        ((Dog)dog).Sleep();
        ((Dog)dog).Eat();
    }
}
```

```
Zzzz
고양이가 먹는 뇸뇸
------------------
Zzzz
우걱우걱
------------------
고양이가 잔다 골골
고양이가 먹는 뇸뇸
------------------
멍멍이가 잔다 드르렁
멍멍이가 먹는 냠냠
```

참고사항

- override 키워드 사용 시 : 반환형, 매개변수의 형태 모두 동일해야함.
- new 키워드 사용 시 : 반환형 달라도 됨.
  - 매개변수의 형태(개수/타입)이 다른 경우는 메소드 오버로딩이 됨.

```
참조 3개
class Dog : Pet
{
    //public override void Sleep() //오버라이딩 불가능
    참조 1개
    public new void Sleep() //하이딩만 가능
    {
        Console.WriteLine("멍멍이가 잔다 드르렁");
    }

    참조 1개
    public new bool Eat() //하이딩도 가능
    {
        Console.WriteLine("멍멍이가 먹는 냠냠");
        return true;
    }
}
```

추상 클래스

인스턴스를 생성할 수 없는 클래스

abstract class 추상 클래스

- 인스턴스를 가질 수 없는 클래스
  - 구현은 할 수 있지만 인스턴스를 만들 수 없다.
  - abstract class의 derived class만이 인스턴스를 만들 수 있다.
- 추상 메소드를 가질 수 있다.
  - 일반적인 class의 멤버 (필드, 프로퍼티, 메소드, ..)도 정의할 수 있지만…

```
참조 6개
class Animal { }
참조 1개
class Human : Animal { }
참조 1개
class Dog : Animal { }

참조 0개
internal class Program
{
    참조 0개
    static void Main(string[] args)
    {
        Animal animal = new Animal();
        Animal human = new Human();
        Animal dog = new Dog();
    }
}
```

```
참조 6개
abstract class Animal { }
참조 1개
class Human : Animal { }
참조 1개
class Dog : Animal { }

참조 0개
internal class Program
{
    참조 0개
    static void Main(string[] args)
    {
        Animal animal = new Animal();
        Animal human = new Human();
        Animal dog = new Dog();
    }
}
```

abstract method (추상 메소드)

- abstract class만이 가질 수 있는 메소드
- 메소드의 header만 정의하고 body를 만들 수 없다.
- abstract method의 body는 반드시 파생 클래스에서 구현해야 하는 책임을 갖는다.
  - 구현하지 않으면 에러가 발생한다. 즉, 강제로 구현할 수 밖에 없다.
  - 파생 클래스로 구현한 인스턴스는 해당 메소드가 있음을 보장한다.
  - 파생 클래스에서 구현할 때 override 키워드를 반드시 사용해야 한다.
- 파생 클래스에서 구현을 해야 하기 때문에 private 접근제한자를 사용할 수 없다.

abstract method (추상 메소드)

```
참조 4개
abstract class Animal
{
    private int _age;
    참조 0개
    public int Age { get; set; }
    참조 0개
    public abstract void Sleep();
}
참조 1개
class Human : Animal { }
참조 1개
class Dog : Animal { }

참조 0개
internal class Program
{
    참조 0개
    static void Main(string[] args)
    {
        //Animal animal = new Animal();
        Animal human = new Human();
        Animal dog = new Dog();
    }
}
```

```
참조 4개
abstract class Animal {
    private int _age;
    참조 0개
    public int Age { get; set; }
    참조 4개
    public abstract void Sleep();
}
참조 1개
class Human : Animal {
    참조 3개
    public override void Sleep() {
        Console.WriteLine("인간잔다.");
    }
}
참조 1개
class Dog : Animal {
    참조 3개
    public override void Sleep() {
        Console.WriteLine("개잔다.");
    }
}

참조 0개
internal class Program
{
    참조 0개
    static void Main(string[] args)
    {
        //Animal animal = new Animal();
        Animal human = new Human();
        Animal dog = new Dog();
        human.Sleep();
        dog.Sleep();
    }
}
```

Animal의 파생클래스인 Human과 Dog 인스턴스는 모두 Sleep() 메소드가 구현되어 있다.

override(재정의)했기 때문에 Animal 타입 상태에서 호출해도 파생 클래스의 재정의 메소드가 실행된다.

인터페이스

극단적인 추상 클래스

interface 인터페이스

- 인터페이스 선언

```
interface 인테페이스_이름
{
    반환형 메소드이름a();
    반환형 메소드이름b();
}
```

- 특징
  - 클래스/구조체와 비슷하지만 메소드, 프로퍼티, 인덱서, 이벤트만 멤버로 갖을 수 있다. (즉, 필드를 갖을 수 없다.)
  - 메소드, 프로퍼티 등은 구현을 할 수 없다. (일종의 추상 멤버)
  - 접근 제한자를 사용할 수 없다. (모두 public)
  - 인스턴스를 만들 수 없다. (인터페이스를 상속하는 파생클래스에서 만들어야 한다.)
    - 파생 클래스에서는 인터페이스에 선언한 모든 (추상의 성격을 갖는) 메소드와 프로퍼티를 구현.
  - 참조는 갖을 수 있다. (추상 클래스와 동일)
  - 파생클래스 입장에서 기본 클래스와 다르게 인터페이스는 n개 상속할 수 있다.
    - 다중 상속 흉내를 낼 수 있다.
  - 인터페이스 이름은 일반적으로 I로 시작한다.

interface 인터페이스

```
참조 4개
interface IPrintable
{
    참조 3개
    void Print(string data);
}

참조 1개
class SoftData : IPrintable
{
    참조 2개
    public void Print(string data)
    {
        Console.WriteLine(data);
    }
}

참조 1개
class HardData : IPrintable
{
    참조 2개
    public void Print(string data) {
        using(var file = File.CreateText("a.txt")){
            file.WriteLine(data);
        }
    }
}
```

interface 인터페이스

```
참조 0개
internal class Program
{
    참조 0개
    static void Main(string[] args)
    {
        IPrintable[] printables = new IPrintable[2];
        printables[0] = new SoftData();
        printables[1] = new HardData();

        for(int i=0; i < printables.Length; i++) {
            var data = "나는 글씨를 남길거야:" + ((i + 1) * 1000);
            printables[i].Print(data);
        }

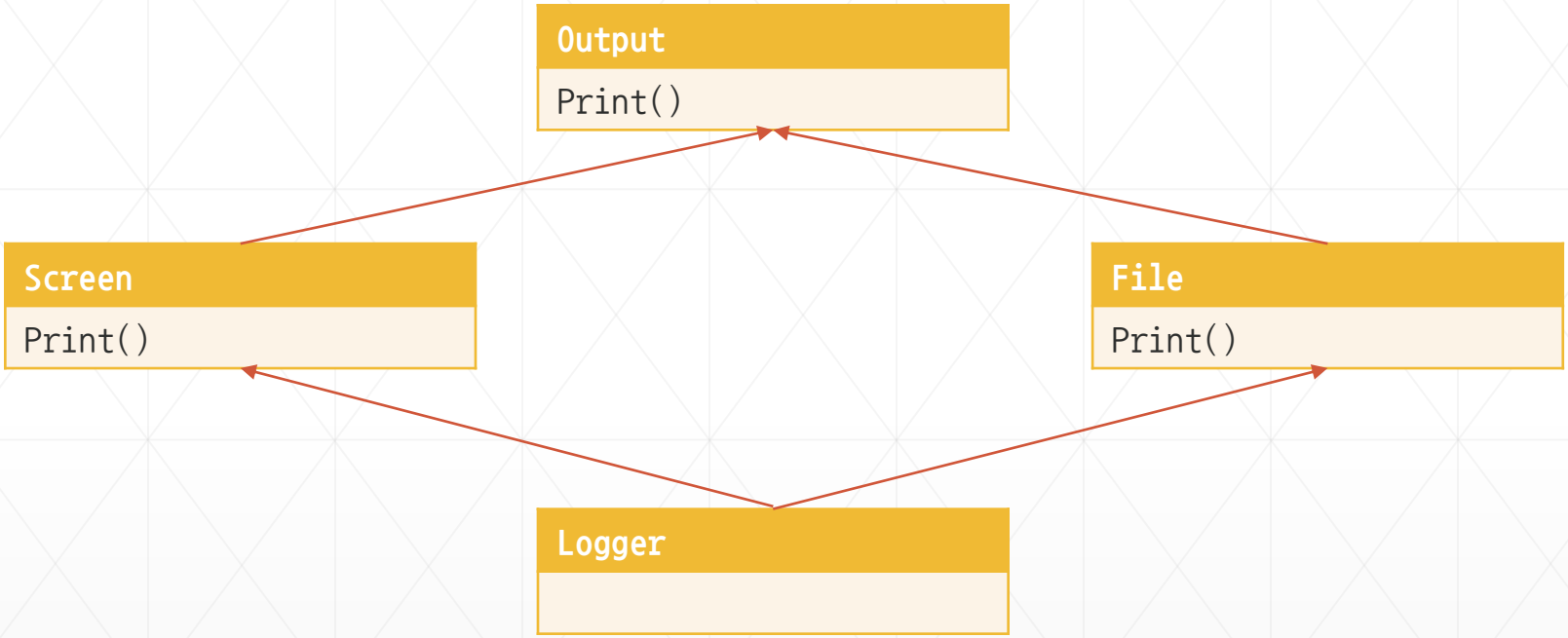
    }
}
```

interface 인터페이스

- 인터페이스 의미
  - 파생 클래스가 어떤 메소드(또는 프로퍼티 또는,...)를 반드시 구현해야 하는지 또는 구현되어 있는지를 보증하는 것
  - 인터페이스는 내용이 아닌 외형만을 물려주는 것.
  - 상속을 통해 구현을 물려주는 것이 아니라 다양한 형태 (다형성)의 의미를 갖출 수 있게 할 때 사용

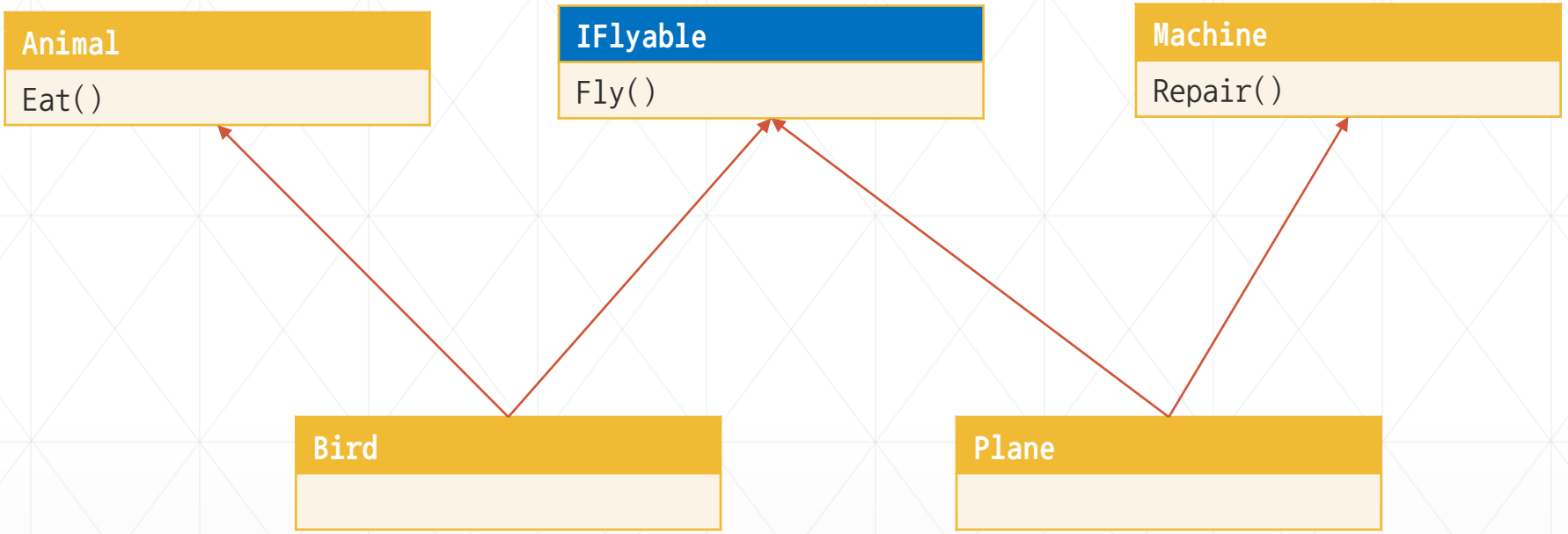
다중 상속

- The Deadly Diamond of Death (죽음의 다이아몬드)



- Logger logger = new Logger();
- logger.Print(); // Screen의 Print()? Output의 Print()?

interface 활용 예시




- ```
  Iflyable bird = new Bird();
  IFlyable plane = new Plane();
  bird.Fly();
  plan.Fly();
  ((Animal)bird).Eat();
  ((Machine)plan).Repair();
  ```
  ```