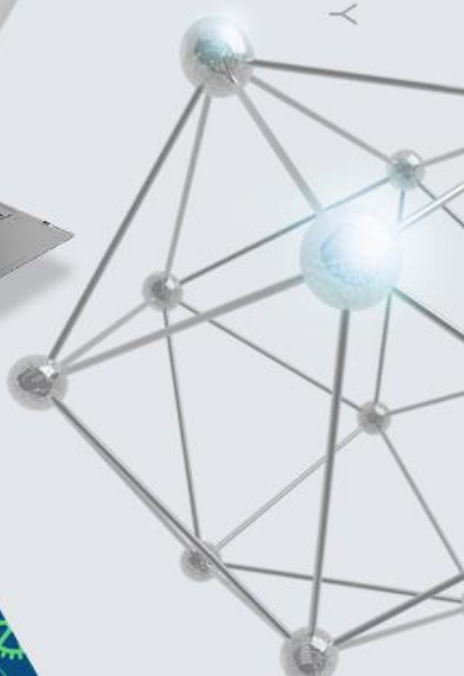
한국기술교육대학교
온라인평생교육원

The 4th Industrial Revolution is characterized by super connectivity and super intelligence, where various products and services are connected to the network, and artificial intelligence and information communication technologies are used in 3D printing, unmanned transportation, robotics, Of the world's most advanced technologies.

TECHNOLOGY

C# 프로그래밍

파생 클래스와 인터페이스



The 4th Industrial Revolution is characterized by super connectivity and super intelligence, where various products and services are connected to the network, and artificial intelligence and information communication technologies are used in 3D printing, unmanned transportation, robotics, Of the world's most advanced technologies.

파생 클래스와 인터페이스

학/습/목/표

1. 파생 클래스를 이해하고 설명할 수 있다.
2. 인터페이스를 이해하고 구현할 수 있다.
3. 네임스페이스를 이해하고 구현할 수 있다.

학/습/내/용

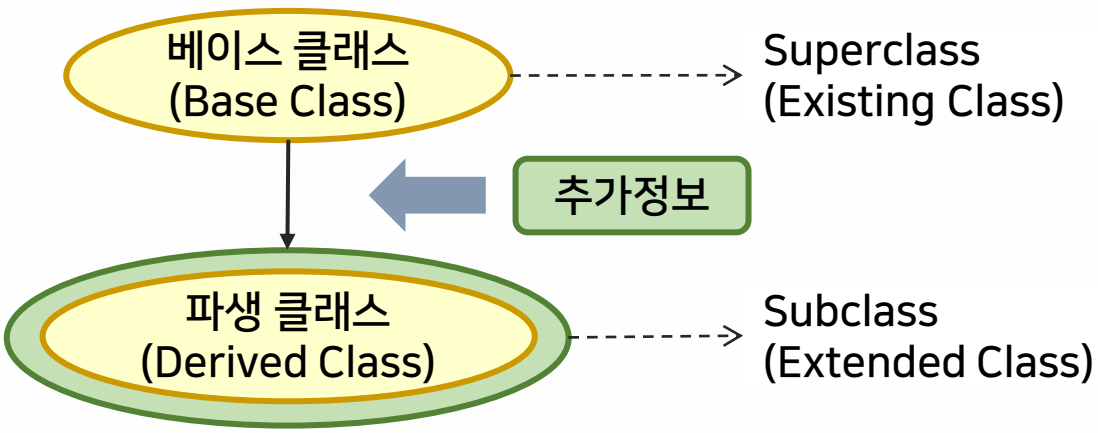
1. 파생 클래스
2. 인터페이스
3. 네임스페이스

1. 파생 클래스

1) 파생 클래스의 개념

(1) 개념

- 파생 클래스 개념



(2) 상속

- 상속(Inheritance)
  → 베이스 클래스의 모든 멤버들이 파생 클래스로 전달 되는 기능임
  → 클래스의 재사용성(Reusability) 증가함

- 상속의 종류
  → 단일 상속 : 베이스 클래스 1개임
  → 다중 상속 : 베이스 클래스 1개 이상임

- C#은 단일 상속만 지원
  → 1개의 베이스 클래스만 사용 가능한 단일 상속만 지원함

2) 파생 클래스의 정의

(1) 형태

- 파생 클래스의 정의 형태

선언 형태

```
[class-modifiers] class DerivedClassName: BaseClassName {
  // member declarations
}
```

- 파생 클래스의 예

예

```
class BaseClass {
  int a;
  void MethodA {
    // ...
  }
}
```

```
class DerivedClass: BaseClass {
  int b;
  void MethodB {
    // ...
  }
}
```

3) 파생 클래스의 필드와 생성자

(1) 파생 클래스의 필드

- 파생 클래스의 필드

→ 클래스의 필드 선언 방법과 동일함

→ 베이스 클래스의 필드명과 다른 경우 : 상속됨

→ 베이스 클래스의 필드명과 동일한 경우 : 숨겨짐

- base 지정어 : 베이스 클래스 멤버 참조

(2) 파생 클래스의 생성자

- 파생 클래스의 생성자

→ 형태와 의미는 클래스의 생성자와 동일함

→ 명시적으로 호출하지 않으면, 기본 생성자가 컴파일러에 의해 자동적으로 호출함

→base( )

-베이스 클래스의 생성자를 명시적으로 호출

→ 실행과정

-필드의 초기화 부분 실행

-베이스 클래스의 생성자 실행

-파생 클래스의 생성자 실행

4) 메소드 재정의, 가상 메소드

(1) 메소드 재정의

- 메소드 재정의(Method Overriding)

→ 베이스 클래스에서 구현된 메소드를 파생 클래스에서 구현된 메소드로 대체함

→ 메소드의 시그너처가 동일한 경우 : 메소드 재정의

→ 메소드의 시그너처가 다른 경우 : 메소드 중복

(2) 가상 메소드

- 가상 메소드(Virtual Method)

→ 지정어 virtual로 선언된 인스턴스 메소드임

→ 파생 클래스에서 재정의해서 사용할 것임을 알려주는 역할을 함

- new 지정어 : 객체 형에 따라 호출
- override 지정어 : 객체 참조가 가리키는 객체에 따라 호출

5) 봉인 메소드와 추상 클래스

(1) 봉인 메소드

- 봉인 메소드(Sealed Method)

→ 수정자가 sealed로 선언된 메소드임

→ 파생 클래스에서 재정의를 허용하지 않음

→ 봉인 클래스 : 모든 메소드는 묵시적으로 봉인 메소드임

(2) 추상 클래스

- 추상 클래스(Abstract Class)

→ 추상 메소드를 갖는 클래스

- 추상 메소드(Abstract Method)

- 실질적인 구현을 갖지 않고 메소드 선언만 있는 경우

- 추상 클래스 선언 방법

선언 형태

```
abstract class AbstractClass {
  public abstract void MethodA();
  void MethodB() {
    // ...
  }
}
```

5) 봉인 메소드와 추상 클래스

(2) 추상 클래스

- 추상 클래스는 구현되지 않고, 단지 외형만을 제공

→ 추상 클래스는 객체를 가질 수 없음

→ 다른 외부 클래스에서 메소드를 일관성 있게 다루기 위한 방법 제공함

→ 다른 클래스에 의해 상속 후 사용 가능함

- abstract 수정자는 virtual 수정자의 의미 포함

→ 추상 클래스를 파생 클래스에서 구현

- override 수정자를사용하여추상메소드를재정의함
- 접근 수정자 항상 일치함

6) 메소드 설계와 클래스형 변환

(1) 메소드 설계

- 메소드를 파생 클래스에서 재정의하여 사용

→ C# 프로그래밍에 유용한 기능임

→ 베이스 클래스에 있는 메소드에 작업을 추가하여 새로운 기능을 갖는 메소드 정의 : base 지정어 사용

6) 메소드 설계와 클래스형 변환

(2) 클래스형 변환

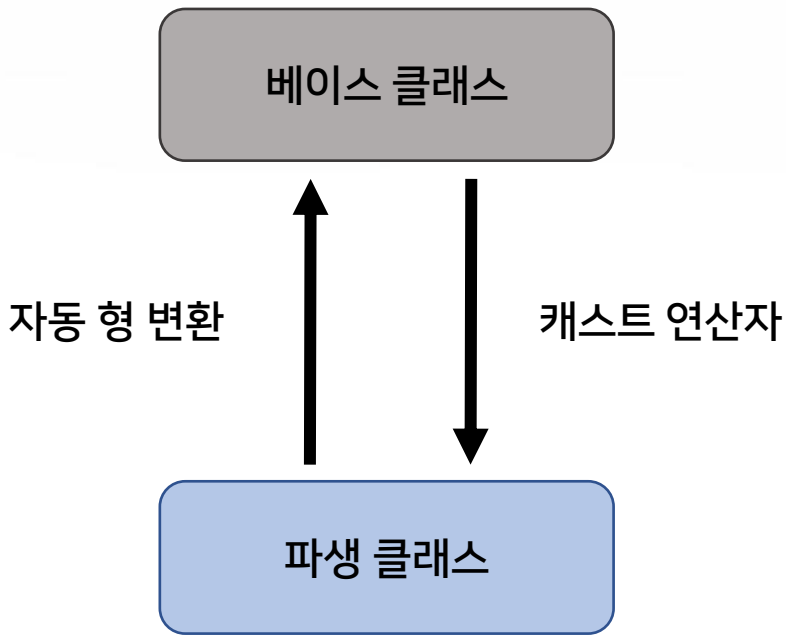
- 클래스형 변환

상향식 캐스트 (캐스팅-업)

→ 타당한 변환
→ 파생 클래스형의 객체가 베이스 클래스형의 객체로 변환함

하향식 캐스트 (캐스팅-다운)

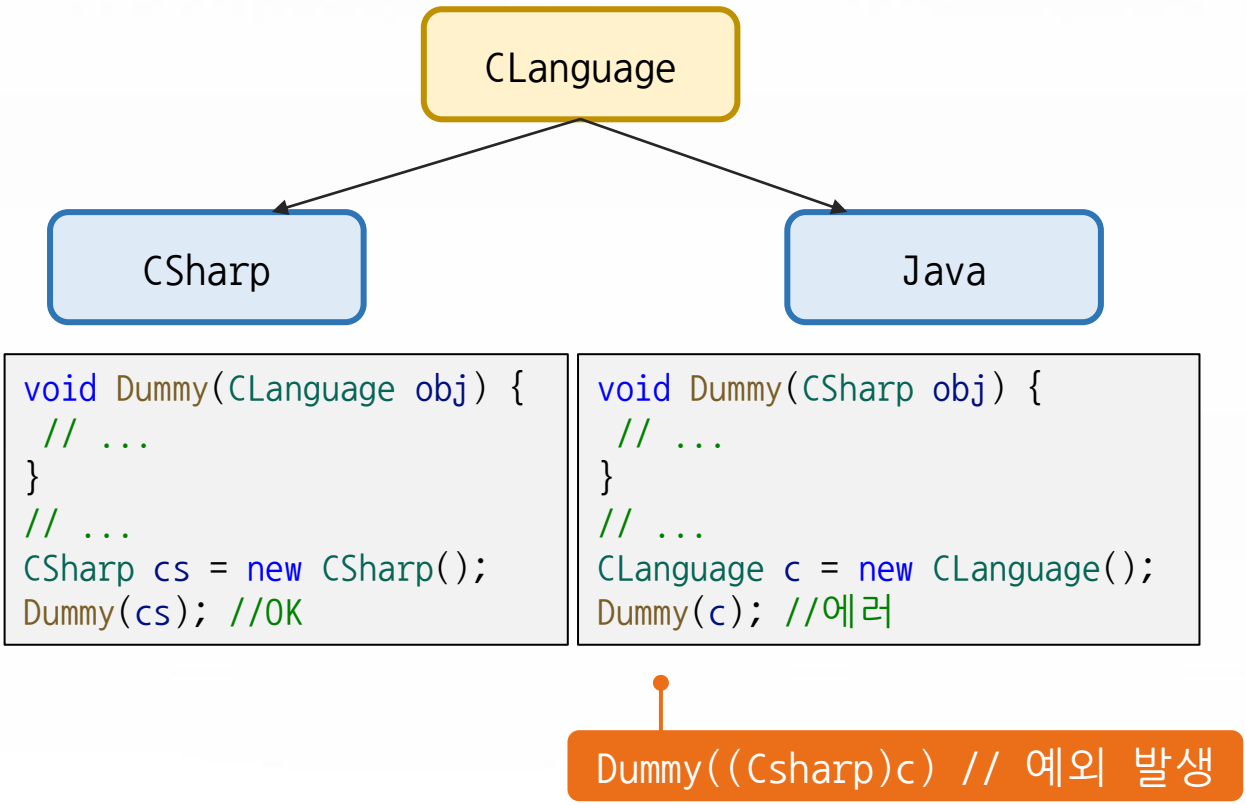
→ 타당하지 않은 변환
→ 캐스트 연산자 사용 : 예외 발생



6) 메소드 설계와 클래스형 변환

(2) 클래스형 변환

- 클래스형 변환의 예



7) 다형성

(1)다형성(Polymorphism)의 의미

- 적용하는 객체에 따라 메소드의 의미가 달라지는 것을 의미함
- C# 프로그래밍 : virtual 과 override의 조합으로 메소드 선언함

```
CLanguage c = new Java();
c.Print();
```

C의 형은 CLanguage이지만
Java 클래스의 객체를 가리킴

2. 인터페이스

1) 인터페이스의 의미와 특징

(1) 의미

- 인터페이스의 의미

→ 사용자 접속을 기술할 수 있는 프로그래밍 단위

→ 구현되지 않은 멤버들로 구성된 수수한 설계의 표현

(2) 특징

- 인터페이스의 특징

→ 지정어 interface 사용함

→ 멤버로는 메소드, 프로퍼티, 인덱스, 이벤트가 올 수 있으며 모두 구현부분이 없음

→ 다중 상속 가능함

→ 접근수정자 : public, protected, internal, private, new

2. 인터페이스

2) 인터페이스의 선언과 확장

(1) 인터페이스의 선언

- 인터페이스의 선언 형태

선언 형태

```
[interface-
modifiers][partial] interface InterfaceNam
e {
 // interface body
}
```

(2) 인터페이스의 확장

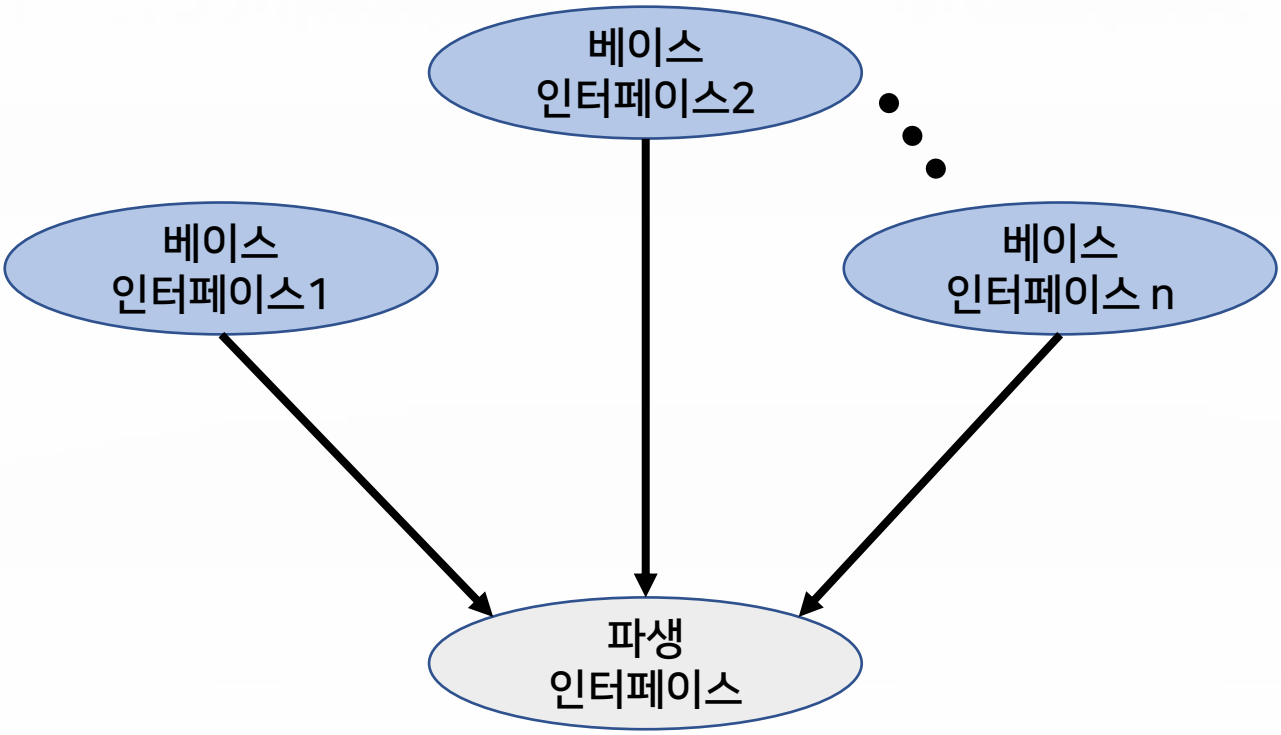
- 인터페이스의 확장 형태

확장 형태

```
[modifiers] interface InterfaceName : List
OfBaseInterfaces {
 // method declarations
 // property declarations
 // indexer declarations
 // event declarations
}
```

3) 인터페이스의 다중 상속

(1) 다중 상속



3) 인터페이스의 다중 상속

(2) 구현 규칙

- 인터페이스 구현 규칙

→ 인테페이스에 있는 모든 멤버는 묵시적으로 public이므로 접근수정자를 public으로 명시함

→ 멤버로는 메소드, 프로멤버 중 하나라도 구현하지 않으면 derived 클래스는 추상 클래스가 됨

(3) 형태

- 인터페이스 구현 형태

구현 형태

```
[class-modifiers] class ClassName : ListOfInterfaces {
  // member declarations
}
```

4) 인터페이스의 구현

(1) 인터페이스 구현

- 클래스 확장과 동시에 인터페이스 구현

선언 형태

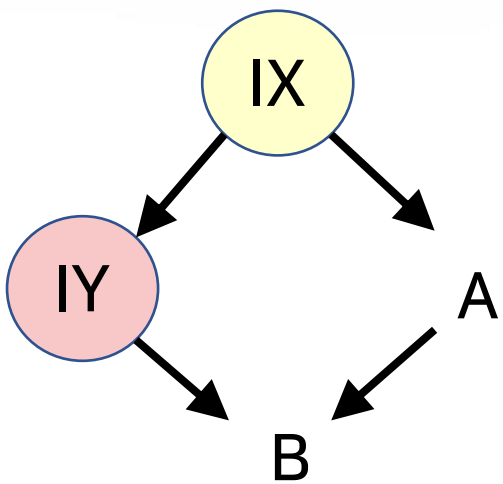
```
[class-modifiers] class ClassName : BaseClass, ListOfInterfaces {
  // member declarations
}
```

(2) 다이아몬드 상속

- 다이아몬드 상속

예

```
interface IX {}
interface IY: IX {}
class A: IX {}
class B: A, IY {}
```



4) 인터페이스의 구현

(2) 다이아몬드 상속

- 예제 : DiamondApp.cs

```
using System;
interface IX { void XMethod(int i); }
interface IY: IX { void YMethod(int i); }
class A: IX {
 private int a;
 public int PropertyA {
  get { return a; } set { a = value; }
 }
 public void XMethod(int i) { a = i; }
}
class B: A, IY {
 private int b;
 public int PropertyB {
  get { return b; }
  set { b = value; }
 }
}
public void YMethod(int i) { b = i; }

class DiamondApp {
 public static void Main() {
  B obj = new B();
  obj.XMethod(5); obj.YMethod(10);
  Console.WriteLine("a = {0}, b = {1}",
  obj.PropertyA, obj.PropertyB);
 }
}
```

5) 인터페이스와 추상 클래스

(1) 인터페이스와 추상 클래스 비교

- 공통점

  → 객체를 가질 수 없음

- 차이점

  → 인터페이스

  - 다중 상속 지원함
  - 오직 메소드 선언만 가능함
  - 메소드 구현 시, override 지정어를 사용할 수 있음

  → 추상 클래스

  - 단일 상속 지원함
  - 메소드의 부분적인 구현 가능함
  - 메소드 구현 시, override 지정어를 사용할 수 없음

3. 네임스페이스

1) 네임스페이스 개요

(1) 정의

- 네임스페이스(Namespace)의 특징

→ 서로 연관된 클래스나 인터페이스, 구조체, 열거형, 델리게이트, 하위 네임스페이스를 하나의 단위로 묶어주는 것

- 예) 여러 개의 클래스와 인터페이스, 구조체, 열거형, 델리게이트 등을 하나의 그룹으로 다루는 수단을 제공함
- 클래스의 이름을 지정할 때 발생 되는 이름 충돌 문제를 해결함

3. 네임스페이스

1) 네임스페이스 개요

(2) 형태

- 네임스페이스 선언 형태

선언 형태

```
namespace NamespaceName {
 // 네임스페이스에 포함할 항목을 정의
}
```

- 네임스페이스 사용

선언 형태

```
using NamespaceName;
// 사용하고자하는 네임스페이스 명시
```

1. 파생 클래스

- C#은 이미 존재하는 클래스에 정보를 추가하여 새로운 클래스를 만들 수 있음
- 이 때 기존의 클래스를 베이스 클래스(Base Class)라 부르며 새로 정의된 클래스를 파생 클래스(Derived Class)라 부름
- 베이스 클래스의 모든 멤버들이 파생 클래스로 옮겨지는 특성을 상속 (Inheritance)이라 부름
- 베이스 클래스로부터 메소드를 상속받을 때, 파생 클래스 내에 같은 이름의 메소드가 있는 경우에 시그너처가다르면중복(Overloading)이 되고 같으면 재정의 (Overriding) 됨

2. 인터페이스

- C#에서 인터페이스란 사용자 접속을 기술할 있는 프로그래밍 단위로, 다시 말해서 구현되지 않은 멤버들로 구성된 순수한 설계의 표현임
- 파생 인터페이스는 여러 개의 베이스 인터페이스를 가질 수 있으며 이와 같이 여러 개의 인터페이스로부터 상속받는 것을 다중 상속이라 부름

3. 네임스페이스

- 네임스페이스란 서로 연관된 클래스나 인터페이스, 구조체, 열거형, 델리게이트, 하위 네임스페이스를 하나의 단위로 묶는 방법