C# 프로그래밍 입문

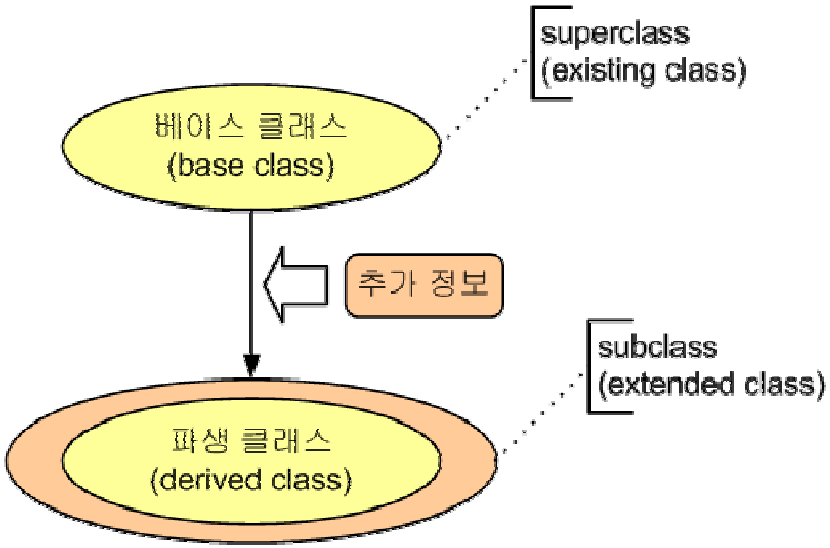
5. 파생클래스와 인터페이스

목차

- 파생 클래스
- 인터페이스
- 네임스페이스

파생 클래스 [1/3]

- 파생 클래스 개념



- 상속(inheritance)
  - 베이스 클래스의 모든 멤버들이 파생 클래스로 전달 되는 기능
  - 클래스의 재사용성(reusability) 증가
- 상속의 종류
  - 단일 상속
    - 베이스 클래스 1개
  - 다중 상속
    - 베이스 클래스 1개 이상
- C#은 단일 상속만 지원

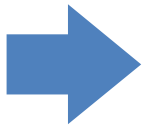
파생 클래스 [2/3]

- 파생 클래스의 정의 형태

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

- 파생 클래스 예

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



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

파생 클래스 [3/3]

- 파생 클래스의 필드
  - 클래스의 필드 선언 방법과 동일
  - 베이스 클래스의 필드명과 다른 경우 - 상속됨
  - 베이스 클래스의 필드명과 동일한 경우 - 숨겨짐
    - base 지정어 – 베이스 클래스 멤버 참조 - 예제 HiddenNameApp
- 파생 클래스의 생성자
  - 형태와 의미는 클래스의 생성자와 동일
  - 명시적으로 호출하지 않으면, 기본 생성자가 컴파일러에 의해 자동적으로 호출 – 예제 DerivedConstructorApp
  - base()
    - 베이스 클래스의 생성자를 명시적으로 호출 – 예제 BaseCallApp
  - 실행과정
    - 필드의 초기화 부분 실행
    - 베이스 클래스의 생성자 실행
    - 파생 클래스의 생성자 실행

메소드 재정의

- 메소드 재정의(method overriding)
  - 베이스 클래스에서 구현된 메소드를 파생 클래스에서 구현된 메소드로 대체
  - 메소드의 시그네처가 동일한 경우 – 메소드 재정의
  - 메소드의 시그네처가 다른 경우 – 메소드 중복
  - 예제 OverridingAndOverloadingApp

가상 메소드 / 봉인 메소드

- 가상 메소드(virtual method)
  - 지정어 virtual로 선언된 인스턴스 메소드
  - 파생 클래스에서 재정의해서 사용할 것임을 알려주는 역할
    - new 지정어 – 객체 형에 따라 호출
    - override 지정어 – 객체 참조가 가리키는 객체에 따라 호출

- 봉인 메소드(sealed method)
  - 수정자가 sealed로 선언된 메소드
  - 파생 클래스에서 재정의를 허용하지 않음
  - 봉인 클래스 – 모든 메소드는 묵시적으로 봉인 메소드

추상 클래스 [1/2]

- 추상 클래스(abstract class)
  - 추상 메소드를 갖는 클래스
    - 추상 메소드(abstract method)
      - 실질적인 구현을 갖지 않고 메소드 선언만 있는 경우

- 추상 클래스 선언 방법

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

추상 클래스 [2/2]

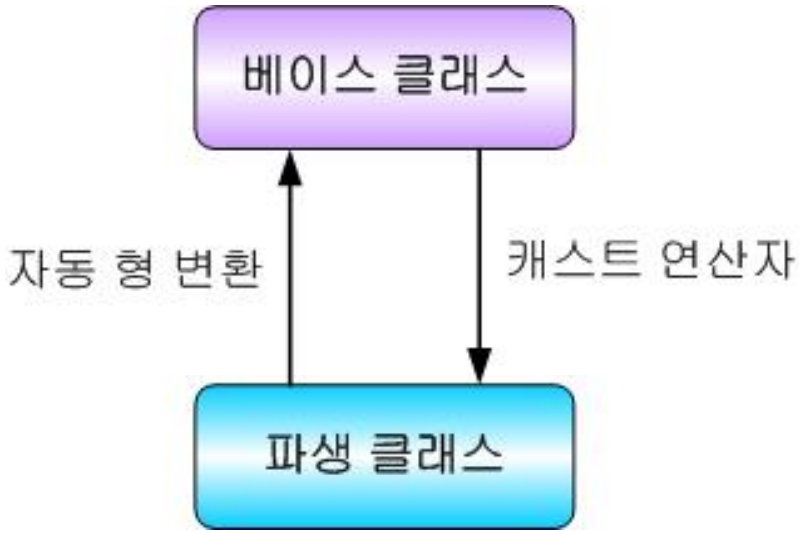
- 구현되지 않고, 단지 외형만을 제공
  - 추상 클래스는 객체를 가질 수 없음
  - 다른 외부 클래스에서 메소드를 일관성 있게 다루기 위한 방법 제공
  - 다른 클래스에 의해 상속 후 사용 가능

- abstract 수정자는 virtual 수정자의 의미 포함
  - 추상 클래스를 파생 클래스에서 구현
    - override 수정자를 사용하여 추상 메소드를 재정의
    - 접근 수정자 항상 일치

메소드 설계

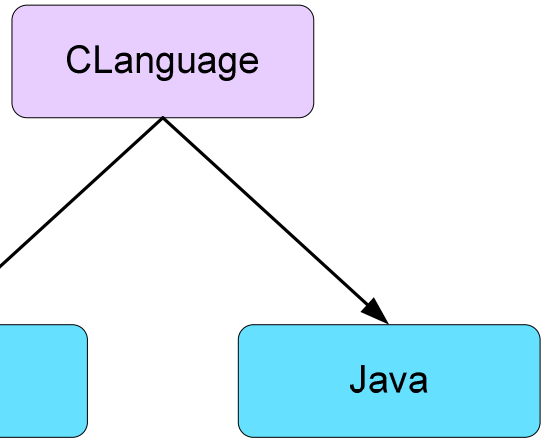
- 메소드를 파생 클래스에서 재정의하여 사용
  - C# 프로그래밍에 유용한 기능
  - 베이스 클래스에 있는 메소드에 작업을 추가하여 새로운 기능을 갖는 메소드 정의 – base 지정어 사용
  - 예제 AddendumApp – 파생 클래스에서 기능을 추가하여 재정의한 프로그램 예제

클래스형 변환 [1/2]



- 상향식 캐스트 (캐스팅-업) – 타당한 변환
  - 파생 클래스형의 객체가 베이스 클래스형의 객체로 변환
- 하향식 캐스트 (캐스팅-다운) – 타당하지 않은 변환
  - 캐스트 연산자 사용 – 예외 발생

클래스형 변환 [2/2]




```
void Dummy(CLanguage obj) {
    // ...
}
// ...
CSharp cs = new CSharp();
Dummy(cs);        // OK
```

```
void Dummy(CSharp obj) {
    // ...
}
// ...
CLanguage c = new CLanguage();
Dummy(c);        // 에러
```

dummy((CSharp)c) // 예외 발생

다형성

- 다형성(polymorphism)
  - 적용하는 객체에 따라 메소드의 의미가 달라지는 것
  - C# 프로그래밍 – virtual 과 override의 조합으로 메소드 선언
    – 예제 VirtualAndOverrideApp

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

c의 형은 CLanguage이지만
Java 클래스의 객체를 가리킴

인터페이스 [1/3]

- 인터페이스의 의미
  - 사용자 접속을 기술할 수 있는 프로그래밍 단위.
  - 구현되지 않은 멤버들로 구성된 수수한 설계의 표현.

- 인테페이스의 특징
  - 지정어 interface 사용.
  - 멤버로는 메소드, 프로퍼티, 인덱스, 이벤트가 올 수 있으며 모두 구현부분이 없음.
  - 다중 상속 가능.
  - 접근수정자 : public, protected, internal, private, new

인터페이스 [2/3]

- 인터페이스 선언 형태

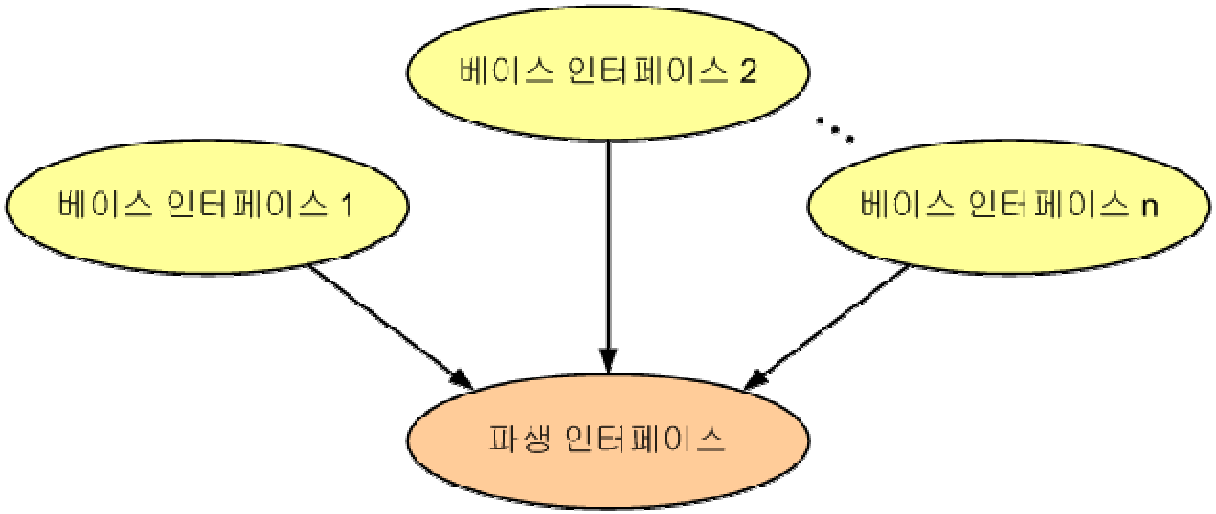
```
[interface-modifiers] [partial] interface InterfaceName {
        // interface body
}
```

- 인터페이스 확장 형태

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

인터페이스 [3/3]

- 인터페이스의 다중 상속



인터페이스 구현 [1/5]

- 인터페이스 구현 규칙
  - 인테페이스에 있는 모든 멤버는 묵시적으로 public이므로 접근수정자를 public으로 명시.
  - 멤버 중 하나라도 구현하지 않으면 derived 클래스는 추상클래스가 됨.

- 인테페이스 구현 형태

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

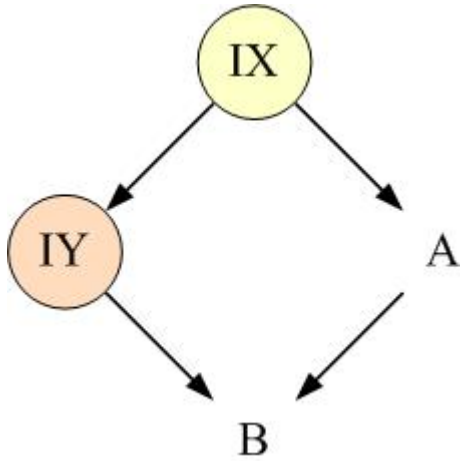
인터페이스 구현 [2/5]

- 클래스 확장과 동시에 인터페이스 구현

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

- 다이아몬드 상속

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



인터페이스 구현 [3/5]

[예제 - ImplementingInterfaceApp.cs]

```
using System;
interface IRectangle {
    void Area(int width, int height);
}
interface ITriangle {
    void Area(int width, int height);
}
class Shape : IRectangle, ITriangle {
    void IRectangle.Area(int width, int height) {
        Console.WriteLine("Rectangle Area : "+width*height);
    }
    void ITriangle.Area(int width, int height) {
        Console.WriteLine("Triangle Area : "+width*height/2);
    }
}
```

인터페이스 구현 [4/5]

[예제 - ImplementingInterfaceApp.cs] – [계속]

```
class ImplementingInterfaceApp {
    public static void Main() {
        Shape s = new Shape();
        // s.Area(10, 10);                    // error - ambiguous !!!
        // s.IRectangle.Area(10, 10);         // error
        // s.ITriangle.Area(10, 10);          // error
        ((IRectangle)s).Area(20, 20);         // 캐스팅-업
        ((ITriangle)s).Area(20, 20);          // 캐스팅-업
        IRectangle r = s;                     // 인터페이스로 캐스팅-업
        ITriangle t = s;                      // 인터페이스로 캐스팅-업
        r.Area(30, 30);
        t.Area(30, 30);
    }
}
```

실행 결과 :

```
Rectangle Area = 400
Triangle Area = 200
Rectangle Area = 900
Triangle Area = 450
```

인터페이스 구현 [5/5]

[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);
    }
}
```

실행 결과 :

```
a = 5, b = 10
```

인터페이스와 추상 클래스

- 공통점
  - 객체를 가질 수 없음
- 차이점
  - 인터페이스
    - 다중 상속 지원
    - 오직 메소드 선언만 가능
    - 메소드 구현 시, override 지정어를 사용할 수 있음
  - 추상 클래스
    - 단일 상속 지원
    - 메소드의 부분적인 구현 가능
    - 메소드 구현 시, override 지정어를 사용할 수 없음

네임스페이스

- 네임스페이스(namespace)
  - 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 제공
    - 클래스의 이름을 지정할 때 발생 되는 이름 충돌 문제 해결
- 네임스페이스 선언

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

- 네임스페이스 사용

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