생성자와 종료자1, 2

2024년 2월 24일 토요일 오후 2:38

- 생성자(Constructor) : 설계된 클래스의 객체를 생성하는 메소드와 같은 것
- 생성자 형식
  - 한정자 클래스 명 (매개변수 명)
    {
    …
    }
- 클래스를 선언할 때 생성자를 구현하지 않았을 경우에는 컴파일러에서 기본 생성자를 만들어준다.
- 생성자의 이름은 클래스의 이름과 같다.
- 종료자 : 클래스의 이름에 ~를 붙여서 사용한다.
  - 생성자와 달리 한정자도 사용하지 않는다.
  - 매개변수도 없다.
  - 오버로딩도 불가능하다
  - 직접 호출할 수도 없다.
  - CLR의 가비지 컬렉터가 객체가 소멸되는 시점을 판단해서 종료자를 호출한다.
  - 종료자를 직접 구현하여 작성할 필요는 없다.

- C# Application

CLR
.Net Framework
Window, O/S
H/W

  - CLR (Common Language Runtime)
    - 원시코드를 컴파일하고나서 컴파일된 코드인 중간언어 IL을 해당 O/S가 최적화하여 읽을 수 있도록 네이티브 코드로 실시간으로 컴파일해준다.
    - JIT
- C# 6.0에서는 포맷팅을 할 때 $ 기호를 이용하여 표현할 수 있도록 하고 있다.

static 필드와 static 메소드

2024년 2월 24일 토요일 오후 3:07

- 한 프로그램 안에는 클래스는 하나만 존재하지만 인스턴스는 여러 개가 존재한다.
- 한 프로그램 안에 똑같은 클래스는 두개가 존재할 수 없다.
- static (정적) 메소드와 정적 필드
    - static 필드나 static 메소드는 클래스에 소속된 것이므로 프로그램 안에 유일하게 존재한다는 의미
    - 클래스명을 이용해서 필드와 메소드를 호출하면 된다.

깊은 복사의 이해

2024년 2월 24일 토요일 오후 3:20

- Value Type (Stack)
- Ref Type (Heap)
- 객체의 인스턴스를 생성하면 Stack 메모리 영역에 존재하게 된다.
    - 인스턴스의 멤버변수는 heap 영역에 존재하게 된다.
    - 객체의 인스턴스에 단순히 다른 인스턴스를 대입 연산을 해주게 되면
      인스턴스는 Stack 메모리 영역에 있지만
      인스턴스의 멤버변수는 동일한 Heap 영역을 가리키게 된다.
      이것이 얕은 복사이다.
    - 깊은 복사는 인스턴스를 반환하는 메소드를 만들어서 반환해주거나 인터페이스를 사용하여 복사할 수 있다.

this, this() 생성자

2024년 2월 24일 토요일 오후 3:37

- this : 객체가 자신을 지칭할 때 사용하는 키워드 입니다.
- this() 생성자
    - 중복되는 생성자 내 코드를 상속받아서 사용

접근 제한자(Access Modifier) 1, 2

2024년 2월 24일 토요일 오후 5:54

- 접근 제한자 / 한정자 (Access Modifier)
  - public :
    - 클래스의 내부 또는 외부 모든 곳에서 접근할 수 있도록 하는 지정자
  - private :
    - 클래스 외부에서는 접근할 수 없도록 하는 지정자.
    - 즉, 내부에서만 접근이 가능하도록 하는 지정자
    - 상속받은 자식(파생) 클래스에서도 접근이 허용이 안됨.
  - protected :
    - 클래스 외부에서는 접근할 수 없도록 하는 지정자.
    - 하지만, 파생 클래스에서는 접근이 가능하도록 하는 지정자.
  - internal :
    - 동일 어셈블리에 있는 코드에서만 public 접근할 수 있는 지정자이다.
    - 다른 어셈블리에 있는 코드에서는 private와 같은 접근 수준을 갖는다.
  - protected internal :
    - 동일 어셈블리에 있는 코드에서만 protected로 접근할 수 있는 지정자.
    - 다른 어셈블리에 있는 코드에서는 private와 같은 접근 수준을 갖는다.
    - main 함수 내에서는 접근 가능함 (static 함수, 같은 코드 파일 내 있는 경우에서 확인)
  - private protected :
    - 동일 어셈블리에 있는 클래스에서 상속받은 클래스 내부에서만 접근이 가능한 지정자.
- 클래스 멤버에 한정자가 지정되지 않았을 경우 무조건 private으로 자동 지정된다.

상속, base 키워드와 sealed한정자

2024년 2월 24일 토요일 오후 10:09

- 부모 클래스 (기반 클래스)
- 자식 클래스 (파생 클래스)
- 상속 클래스 생성/소멸 순서
    - 부모 클래스 생성
    - 자식 클래스 생성
    - 자식 클래스 소멸
    - 부모 클래스 소멸
- 기반 클래스의 인자 생성자에게 파생 클래스가 인자를 전달하는 방법
    - base 키워드를 사용하여 부모 클래스에 생성자, 메소드 등을 사용할 수 있음.
- base 키워드는 기반 클래스를 가리키는 키워드이다.
    - 기반 클래스에 멤버를 접근할 때 사용한다.
- sealed 키워드(한정자)를 사용하면 상속을 봉인하게 되어 상속이 불가능하도록 클래스를 선언할 수 있음.

상속관계의 클래스 형변환

2024년 2월 24일 토요일 오후 10:31

- 파생클래스의 인스턴스는 부모 클래스의 인스턴스로 사용될 수 있다.
- 기반 클래스는 상속받은 클래스로 형변환 할 수 있음
- 상속 관계에 있는 클래스 들 간에는 형변환이 자유로움
- 파생된 클래스를 가리키는 클래스에서 특정 메소드를 만들어야 하는 경우 파생 클래스 별로 해당 메소드를 오버로딩 해야하는 문제가 생김
    - 기반 클래스를 인자로 받아서 형변환을 통해서 처리할 수 있음

is연산자와 as 연산자

2024년 2월 24일 토요일 오후 10:43

- as is 연산자
- as :
    - 형변환(캐스팅)과 같은 역할을 하는 연산자이다.
    - 형변환에 실패했을 경우에는 null 값을 리턴한다.
- is :
    - 해당 객체의 Type(형)이 일치하는 지 여부를 bool 값으로 반환하는 연산자이다.

오버라이딩(virtual, override, new)

2024년 2월 24일 토요일 오후 10:52

- 오버라이드 :
  - 메소드 재정의
- virtual :
  - base(부모) 클래스 메소드 앞에 붙는 키워드
  - derived(파생, 자식) 클래스에 의해서 재정의될 수 있다는 의미를 갖는다.
- override :
  - 자식(derived) 클래스 메소드 앞에 붙는 키워드
  - 부모로부터 받은 메소드를 재정의 한다라는 의미
  - 재정의할 때는 부모의 메소드 이름과 같아야 한다.
  - 프로토타입도 일치해야 한다.
- virtual로 지정된 메소드는 override라는 키워드로 지정된 메소드로 재정의된다.
  생략되는 경우 warning 출력
- new :
  - 오버라이딩과는 다른 개념으로 메소드를 숨기는 기능이다.
  - base 클래스와 별개로 독립적으로 사용
  - Base 클래스로 Derived 클래스 생성 후 new 키워드로 선언된 동일한 메소드라도 파생 클래스의 메소드가 숨겨져서 호출되지 않고 기반 클래스의 메소드가 호출된다.

구조체, 튜플

2024년 2월 25일 일요일 오전 10:17

- C#에서는 struct를 사용하면 Value Type을 만들고, class 사용하면 Reference Type
- Value Type :
  - int, double, float, bool 타입들을 기본타입(Primitive Type)이라고 하는데, struct로 정의된 Value Type이다
  - Value Type은 상속될 수 없으며, 주로 간단한 데이터 값을 저장하는데 사용된다.
- Reference Type :
  - class를 정의해서 만들고, 상속이 가능하다
  - 좀 더 복잡한 데이터와 기능을 정의하는 곳에 많이 사용된다.

- 구조체 :
  - struct라는 키워드를 이용해서 정의한다.
  - 클래스와 같이 메소드, 속성(프로퍼티)등 거의 비슷한 구조를 가지고 있지만, 상속을 할 수 없다.
  - 클래스와 마찬가지로 인터페이스(interface) 구현을 할 수 있다.
  - 구조체는 매개변수가 없는 생성자는 선언할 수 없다.
  - 모든 구조체는 System.Object 형싱글 상속하는 System.ValueType으로부터 직접 상속받음
    - ToString 메소드 등을 사용할 수 있음. (Override 가능)
  - 구조체는 클래스와 달리 ValueType이기 때문에 대입 연산하여 값 복사 시 Deep Copy (깊은 복사)가 이뤄진다.
- Tuple(튜플) :
  - 여러 개의 필드를 담을 수 있는 구조체와 같다.
  - C# 7.0 이전 버전에서는 메소드에서 하나의 값만을 리턴할 수 있었지만, 이후 버전에서는 튜플을 이용해서 복수 개의 값들을 리턴할 수 있게 되었다.
  - Tuple 선언 예
    - var t = (100, 200) ---> Unnamed Tuple // (item1, item2)
    - 튜플은 ( ) 안에 여러 개의 필드를 지정하여 만들 수 있다.
    - var t (Name : "홍길동", Id = "1212") ---> Named Tuple

튜플 리턴 타입을 이용한 메소드 사용하기

2024년 2월 25일 일요일 오전 10:38

- (타입 변수명1, 타입 변수명2, 타입 변수명3) 메소드명(타입 매개변수명1)
  {
  ....
  }
- var result = GetAverage(dataList)
  - result.변수명1, result.변수명2, result.변수명3
  - result.Item1, result.Item2, result.Item3

인터페이스 이해

2024년 2월 25일 일요일 오전 10:57

- 인터페이스
  - interface 키워드 사용
  - interface 인터페이스명
    {
      반환형식 메소드명(매개변수, ...);
      ...
    }
    - 구현부가 없는 메소드를 추상 메소드라 한다.
  - 인터페이스는 필드를 포함하지 않는다.
    - 이벤트
    - 메소드
    - 프로퍼티 만을 멤버로 갖는다.
  - 인터페이스 모든 멤버는 public 접근 권한으로 지정된다.
    따라서, 접근 제한자를 사용할 수 없다.
  - 인터페이스는 구현부(몸통) 없는 추상 멤버를 갖는다.
  - 인터페이스는 다중 상속이 가능하다.
  - 인터페이스는 다른 인터페이스를 상속 받을 수 있다.
  - 클래스에서도 인터페이스를 상속받을 수 있고, 구조체에서도 인터페이스를 상속받을 수 있다.
  - 관용적으로 접두어에 I를 붙여서 선언한다.
    interface IMyInterface
  - 인터페이스는 인스턴스를 만들 수 없다.
  - **C#은 클래스의 다중상속이 불가능하다.**

추상 클래스 이해

2024년 2월 25일 일요일 오전 11:16

- **C#은 클래스의 다중상속이 불가능하다.**
- 인터페이스 상속 형식
  - interface 자식인터페이스명 : 부모인터페이스명
  - 클래스에서 자식 인터페이스를 상속한 경우, 자식 인터페이스가 부모 인터페이스를 상속하였으므로 클래스 내의 부모 인터페이스의 메소드도 구현되어야 한다.
- 추상 클래스 :
  - 인터페이스와 비슷하지만, 추상클래스는 구현(몸통)부를 가질 수 있다.
  - 인스턴스를 가질 수는 없다.
  - 사용하는 한정자는 abstract와 class를 사용한다
  - 선언 형식
    - abstract class 클래스명
      {
        // 클래스와 동일
      }
  - 추상 클래스는 모든 멤버에 접근 제한자를 사용하지 않을 경우 private 설정된다.
  - 추상 메소드를 지정할 때 abstract 키워드를 사용한다.
  - 추상 메소드의 형식
    - public abstract void A();
  - 추상 메소드는 private 이 될 수 없기 때문에 C# 컴파일러는 public, protected, internal, protected internal 중에 하나로 수식되도록 하고 있다.

프로퍼티의 이해

2024년 2월 25일 일요일 오후 1:28

- C# 은닉화
  - Getter, Setter
  - private 변수에 접근할 때 Getter, Setter를 이용한다.
- 프로퍼티
  - 선언 형식
    - 데이터타입 필드명;
    - 접근제한자 데이터타입 프로퍼티명
      {
      get
      {
      return 필드명
      }
      set
      {
      필드명 = value
      }
      }
    - value는 키워드
    - get, set은 접근자(accessor)
  - DemoClass demo = new DemoClass();
    demo.DemoField = 1;
    Console.WriteLine(demo.DemoField);
  - 프로퍼티를 읽기 전용(get 접근자만 구현)
    쓰기 전용(set 접근자만 구현)으로 정의할 수 있다.

7.0에서 자동 프로퍼티 사용법

2024년 2월 25일 일요일 오후 1:39

- 자동프로퍼티 기능
  - get; set; 형식으로 선언
  - C# 3.0에서 도입된 기능
- C# 7.0
  - C# 7.0부터는 자동프로퍼티 초기값이 필요할 때 생성자에 초기화 코드를 작성해야 하는 불편함을 해소할 수 있도록 초기값을 바로 설정할 수 있다.
  - 변수, 프로퍼티를 분리하는 것이 아니라 public 한정자로 프로퍼티 get, set 형식으로 구현
  - 객체 생성할 때 객체의 필드를 초기화 하는 방법
    - 선언 방법
    - 클래스명 인스턴스명 = new 클래스명()
      {
        프로퍼티1 이름 = 값,
        프로퍼티2 이름 = 값,
        …
      }

인터페이스에서 자동프로퍼티 사용

2024년 2월 25일 일요일 오후 2:52

- 인터페이스에서 자동 프로퍼티는 C# 컴파일러가 자동으로 구현해주지 않는다.
- 따라서, 해당 인터페이스를 상속받는 클래스에서 구현해주어야 한다.

추상클래스에서 자동프로퍼티 사용

2024년 2월 25일 일요일 오후 3:01

- 추상 프로퍼티 :
    - abstract가 붙어 있으면 구현이 안된 프로퍼티
    - 상속받은 클래스에서 override 하여 구현해주어야 한다.

ArrayList 사용하기

2024년 2월 25일 일요일 오후 3:07

- Collection(컬렉션) :
  - 데이터 모음을 담는 자료구조
  - 배열이나 스택, 큐 등을 C#에서는 컬렉션이라는 이름으로 제공
  - .Net 프레임워크에서 사용하는 컬렉션은 ICollection 인터페이스를 상속받는다.
    - 배열은 System.Array 타입이다.
      - array.GetType().BaseType
    - System.Array는 ICollection 인터페이스를 상속받기 때문에 배열은 컬렉션의 일부이다.
  - ArrayList, Queue, Stack, Hashtable
- ArrayList :
  - 배열과 비슷한 컬렉션
  - 배열처럼 []인덱스로 요소에 접근이 가능하고, 특정 요소를 바로 읽고 쓸 수 있다.
  - 하지만, 배열을 선언할 때는 배열 크기를 지정하는 반면에 ArrayList는 크기를 지정하지 않는다.
  - 요소의 추가, 삭제에 따라서 자동으로 크기를 늘였다 줄였다 한다.
  - ArrayList 컬렉션은 모든 타입의 변수를 담을 수 있다.
    (C#에서 제공하는 모든 컬렉션은 모든 타입의 변수를 담을 수 있다. 그 이유는 컬렉션의 요소들은 object 타입으로 지정되기 때문)
  - ArrayList 값 추가/삭제 메소드
    - Add(data) : 값을 뒤에 추가
    - RemoveAt(index) : 리스트의 해당 인덱스를 찾아 제거
    - Remove(data) : 매개변수로 전달된 data를 찾아서 제거 (중복된 값이 존재하는 경우 처음 찾은 값을 제거)
    - Insert(index, data) : index 위치에 data를 삽입
  - 컬렉션의 모든 요소들은 object 타입이기 때문에 foreach문에서 object obj가 가능하다.

Queue, Stack, Hashtable 사용

2024년 2월 25일 일요일 오후 3:26

- Queue :
  - FIFO (First In First Out) 선입선출
  - Enqueue : 데이터 입력
  - Dequeue : 데이터 출력
- Stack :
  - LIFO (Last In First Out) 후입선출
  - push : 데이터 입력
  - pop : 데이터 출력
- Hashtable :
  - 탐색 속도가 빠르다
  - Key
  - Value
  - hashtable["key"] -> value 값 반환
  - hasing 알고리즘을 이용한 데이터 검색이 이루어지는 방식의 자료구조
    - 키를 이용해서 한번에 데이터가 저장되어 있는 컬렉션 내의 주소를 계산해 낸다.

컬렉션 초기화, 인덱서

2024년 2월 25일 일요일 오후 4:27

- 데이터를 초기화 하는 방법
  - 컬렉션 초기자를 이용한 초기화 방법
    - 컬렉션 초기자는 IEnumerable 인터페이스를 상속 받아 Add() 메소드를 구현하고 있다.
    - 컬렉션 초기자는 Stack, Queue 에서는 사용할 수 없다.
      - Stack과 Queue는 Add() 메소드가 없기 때문
    - Collection 타입 변수명 = new Collection 타입() { 값1, 값2, 값3 … }
  - 배열을 이용한 초기화 방법
    - Stack과 Queue에서도 사용할 수 있다.
    - Collection 타입 변수명 = new Collection 타입 (array 변수)
  - Hashtable 초기화할 때 Dictionary Initializer를 이용할 수 있다.
    - 형식1

      ```
      Hashtable 변수명 = new Hashtable()
      {
          [key] = value,
          [key] = value,
          …
      };
      ```

    - 형식2

      ```
      Hashtable 변수명 = new Hashtable()
      {
          {key, value},
          {key, value},
          …
      };
      ```

- 인덱서(Indexer) :
  - 클래스 객체의 데이터를 배열 형태로 인덱스를 사용해서 엑세스할 수 있도록 해주는 것
  - 객체를 마치 배열처럼 사용할 수 있도록 해준다.
  - 인덱스 선언 방법
    - class 클래스명

      ```
      {
          접근제한자 인덱서형식 this[형식 idx]
          {
              get
              {
                  index를 이용해서 네부 데이터를 반환
              }
              set
              {
      ```

```
                index를 이용해서 네부 데이터를 저장
            }
        }
    }
```

yield 키워드

2024년 2월 25일 일요일 오후 4:48

- Enumerator (Iterator) :
  - 집합적인 데이터셋으로부터 데이터를 하나씩 호출자에게 보내주는 기능
- yield 키워드는 호출자에게 컬렉션 데이터를 하나씩 리턴할 때 사용하는 키워드
- yield 사용방식
  - yield return : 컬렉션 데이터를 하나씩 리턴하는데 사용
  - yield break : 리턴을 중지하고 Iteration 루프를 빠져나올 때 사용
- Enumerator :
  - 데이터 요소를 하나씩 리턴하는 기능을 하는 것
  - 위의 기능을 구현하기 위해서는 C#이나 .NET에서는 IEnumerator라는 인터페이스를 구현해야 한다.
  - IEnumerator 는 Current(속성), MoveNext() (메소드), Reset() (메소드) 3개의 멤버로 이루어져 있다.
  - 따라서, Enumerator가 되기 위해서는 Current와 MoveNext() 를 반드시 구현해야 함
  - 컬렉션 클래스는 Enumeration이 가능한 클래스인데, 이러한 클래스들을 Enumerable 클래스라고 한다.
    - Enumerable 클래스는 IEnumerable 인터페이스를 구현해야한다.
    - IEnumerable 인터페이스는 GetEnumerator() 메소드를 하나를 가지고 있다.
    - GetEnumerator() 메소드는 IEnumerator 구현한 객체를 리턴해준다.
- 컬렉션 타입이나 또는 Enumerable 클래스에서 GetEnumerator() 메소드를 구현하는 방법
  - yield 키워드를 사용
  - GetEnumerator() 메소드에서 yield return 사용하면 컬렉션 데이터를 순차적으로 하나씩 넘겨주는 코드를 구현할 수 있고, 리턴타입은 IEnumerator 인터페이스를 리턴할 수 있다.