일반화 프로그래밍

이것이 C#이다

Contents

11.1 일반화 프로그래밍이란?

- 일반화
  - 특수한 개념으로부터 공통된 개념을 찾아 묶는 것
- 일반화 프로그래밍
  - 일반화의 대상 – 데이터 형식
- 내부 논리는 같은데 데이터 형식 때문에 오버로딩을 해야 한다면?

```
void CopyArray( string[] source, string [] target )
{
    for( int i = 0; i < source.Length; i++ )
        target[i] = source[i];
}
```

- 오버로딩 없이 모든 형식을 지원하는 프로그래밍 패러다임

11.2 일반화 메소드

- 데이터 형식을 일반화한 메소드
- 선언 형식

```
한정자 반환 형식 메소드이름<형식 매개 변수> ( 매개 변수 목록 )
{
    // …
}
```

- 메소드 일반화 단계
  - 데이터 형식이 사용된 부분을 T 기호로 치환
  - < >를 이용해 형식을 매개 변수로 넘겨준다.
  - 메소드 호출 시 < > 사이의 T 대신에

```
void CopyArray( T[] source, T[] target )
{
    for( int i = 0; i < source.Length; i++ )
        target[i] = source[i];
}
```

```
int[] source = { 1, 2, 3, 4, 5 };
int[] target = new int[source.Length];

형식 매개 변수 T에 int를 대입합니다.

CopyArray<int>(source, target);

foreach (int element in target)
    Console.WriteLine(element);
```

- 데모 예제 - CopyingArray

11.3 일반화 클래스

- 데이터 형식을 일반화한 클래스
- 선언 형식

```
class 클래스이름 <형식 매개 변수>
{
    // …
}
```

- 클래스 일반화 단계
  - 데이터 형식은 다르나
  - 형식 매개 변수를 이용
  - 객체 생성시 입력 받은

```
class Array_Int
{
class Array_Generic< T >

Array_Generic<int> intArr = new
Array_Generic<double> dblArr =

    public double GetElement( int i
}
```

```
class Array_Generic
{
    private int[] array;
    // …
    public int GetElement( int index ) { return array[index]; }
}
```

```
Array_Generic<int> intArr = new Array_Generic<int>();
```

```
class Array_Generic<T>
{
    private T[] array;
    // …
    public T GetElement( int index ) { return array[index]; }
}
```

```
Array_Generic<double> dblArr = new Array_Generic<double>();
```

```
class Array_Generic
{
    private double[] array;
    // …
    public double GetElement( int index ) { return array[index]; }
}
}
```

- 데모 예제 - Generic

11.4 형식 매개 변수 제약시키기

- 특정 조건을 갖춘 형식에만 대응하는 형식 매개 변수로 제한
- 형식 제약 문법
  - where **형식 매개 변수** : **제약조건**

```
class BaseArray<U> where U : Base
{
    public U[] Array{ get;set;}
    public BaseArray(int size)
    {
        Array = new U[size];
    }

    public void CopyArray<T>(T[] Source) where T : U
    {
        Source.CopyTo(Array, 0);
    }
}
```

- 데모 예제 – ConstraintsOnTypeParameters

11.5 일반화 컬렉션

- 컬렉션은 object 형식에 기반하기 때문에 태생적 성능 문제 내포
  - 일반화 컬렉션으로 해결
  - →컴파일 시 컬렉션에서 사용할 형식 결정
  - →잘못된 형식의 객체를 담게 될 위험 회피
- System.Collections.Generic 네임스페이스
  - List<T>
  - Queue<T>
  - Stack<T>
  - Dictionary<TKey, TValue>

11.5.1 List<T>

- 비일반화 클래스 ArrayList와 동일한 기능과 사용법.
- 차이점
  - 인스턴스 만들 때 형식 매개 변수 필요
  - 형식 매개 변수로 입 력한 형식 외에는 입력을 허용하지 않음
- 데모 예제 – UsingGenericList

11.5.2 Queue<T>

- 비일반화 클래스인 Queue와 동일한 기능
- 사용 방법 상의 차이점
  - 형식 매개 변수를 요구
- 데모 예제 - UsingGenericQueue

11.5.3 Stack<T>

- 비일반화 클래스인 Stack과 동일한 기능
- 사용 방법 상의 차이점
  - 형식 매개 변수 요구
- 데모 예제 – UsingGenericStack

11.5.4 Dictionary<TKey, TValue>

- Hashtable의 일반화 버전.
- 사용 방법 상의 차이점
  - 2개의 형식 매개 변수 요구
  - TKey는 Key, TValue는 Value를 위한 형식

- 데모 예제 – UsingDictionary

11.6 foreach를 사용할 수 있는 일반화 클래스

- **IEnumerable과 IEnumerator 상속 구현→ 성능저하**
  - **일반화 클래스를** foreach**에 사용 할 경우**
  - → IEnumerable<T>, IEnumerator<T> **구현**
- **IEnumerable<T>와 IEnumerator<T>의 메소드 및 프로퍼티**

메소드	설명
IEnumerator GetEnumerator( )	IEnumerator 형식의 객체를 반환(IEnumerable로부터 상속받은 메소드)
IEnumerator<T> GetEnumerator( )	IEnumerator<T> 형식의 객체를 반환

메소드 또는 프로퍼티	설명
boolean MoveNext( )	다음 요소로 이동합니다. 컬렉션의 끝을 지난 경우에는 false, 이동이 성공한 경우에는 true를 반환합니다.
void Reset( )	컬렉션의 첫 번째 위치의 "앞"으로 이동합니다. 첫 번째 위치가 0번일 때, Reset( )을 호출하면 –1번으로 이동하는 것이죠. 첫 번째 위치로의 이동은 MoveNext( )를 호출한 다음에 이루어집니다.
Object Current { get; }	컬렉션의 현재 요소를 반환합니다(IEnumerator로부터 상속받은 프로퍼티).
T Current(get; }	컬렉션의 현재 요소를 반환합니다.

- **데모 예제 - EnumerableGeneric**

Thank You !

이것이 C#이다