3) 델리게이트, 람다 및 LINQ

매 강의 강의자료 시작에 PDF파일을 올려두었어요!

▼ PDF 파일

[수업 목표]

- 델리게이트와 람다의 개념과 사용법을 이해한다.
- LINQ (Language Integrated Query)를 이해하고 사용할 수 있다.

[목차]

모든 토글을 열고 닫는 단축키
Windows : `Ctrl` + `alt` + `t`
Mac : `⌘` + `⌥` + `t`

---

01. 델리게이트 (Delegate)

▼ 1) 델리게이트란?

- 델리게이트(delegate)는 메서드를 참조하는 타입입니다.
- 다른 프로그래밍 언어에서는 함수 포인터라는 용어를 사용하기도 합니다.

- 델리게이트를 이용하면 메서드를 매개변수로 전달하거나 변수에 할당할 수 있습니다.

▼ 2) 델리게이트 구현

- **메서드 등록해서 사용하기**

```
delegate int Calculate(int x, int y);

static int Add(int x, int y)
{
    return x + y;
}

class Program
{
    static void Main()
    {
        // 메서드 등록
        Calculate calc = Add;

        // 델리게이트 사용
        int result = calc(3, 5);
        Console.WriteLine("결과: " + result);
    }
}
```

- **하나 이상의 메서드 등록하기**

```
delegate void MyDelegate(string message);

static void Method1(string message)
{
    Console.WriteLine("Method1: " + message);
}

static void Method2(string message)
{
    Console.WriteLine("Method2: " + message);
}

class Program
{
    static void Main()
    {
        // 델리게이트 인스턴스 생성 및 메서드 등록
        MyDelegate myDelegate = Method1;
        myDelegate += Method2;

        // 델리게이트 호출
        myDelegate("Hello!");

        Console.ReadKey();
```

```
    }
}
```

▼ 3) 사용 예제

- 공격 콜백 받기
  - 다음 예제에서는 event 를 붙여서 사용했다
  - event는 할당연산자(=)를 사용할 수 없으며, 클래스 외부에서는 직접 이벤트를 호출할 수 없다.

```
// 델리게이트 선언
public delegate void EnemyAttackHandler(float damage);

// 적 클래스
public class Enemy
{
    // 공격 이벤트
    public event EnemyAttackHandler OnAttack;

    // 적의 공격 메서드
    public void Attack(float damage)
    {
        // 이벤트 호출
        OnAttack?.Invoke(damage);
        // null 조건부 연산자
        // null 참조가 아닌 경우에만 멤버에 접근하거나 메서드를 호출
    }
}

// 플레이어 클래스
public class Player
{
    // 플레이어가 받은 데미지 처리 메서드
    public void HandleDamage(float damage)
    {
        // 플레이어의 체력 감소 등의 처리 로직
        Console.WriteLine("플레이어가 {0}의 데미지를 입었습니다.", damage);
    }
}

// 게임 실행
static void Main()
{
    // 적 객체 생성
    Enemy enemy = new Enemy();

    // 플레이어 객체 생성
    Player player = new Player();

    // 플레이어의 데미지 처리 메서드를 적의 공격 이벤트에 추가
```

```
    enemy.OnAttack += player.HandleDamage;

    // 적의 공격
    enemy.Attack(10.0f);
}
```

02. 람다 (Lambda)

▼ 1) 람다란?

- 람다(lambda)는 익명 메서드를 만드는 방법입니다.
- 람다를 사용하면 메서드의 이름 없이 메서드를 만들 수 있습니다.
- 람다는 델리게이트를 사용하여 변수에 할당하거나, 메서드의 매개변수로 전달할 수 있습니다.

▼ 2) 람다 구현

- 형식

```
(parameter_list) => expression
```

- 정의하기

```
Calculate calc = (x, y) =>
{
    return x + y;
};

Calculate calc = (x, y) => x + y;
```

▼ 3) 사용 예제

```
using System;

// 델리게이트 선언
delegate void MyDelegate(string message);

class Program
{
    static void Main()
    {
```

```
        // 델리게이트 인스턴스 생성 및 람다식 할당
        MyDelegate myDelegate = (message) =>
        {
            Console.WriteLine("람다식을 통해 전달된 메시지: " + message);
        };

        // 델리게이트 호출
        myDelegate("안녕하세요!");

        Console.ReadKey();
    }
}
```

- **게임의 분기 시작을 알리기**

```
// 델리게이트 선언
public delegate void GameEvent();

// 이벤트 매니저 클래스
public class EventManager
{
    // 게임 시작 이벤트
    public event GameEvent OnGameStart;

    // 게임 종료 이벤트
    public event GameEvent OnGameEnd;

    // 게임 실행
    public void RunGame()
    {
        // 게임 시작 이벤트 호출
        OnGameStart?.Invoke();

        // 게임 실행 로직

        // 게임 종료 이벤트 호출
        OnGameEnd?.Invoke();
    }
}

// 게임 메시지 클래스
public class GameMessage
{
    public void ShowMessage(string message)
    {
        Console.WriteLine(message);
    }
}

// 게임 실행
static void Main()
{
    // 이벤트 매니저 객체 생성
```

```
    EventManager eventManager = new EventManager();

    // 게임 메시지 객체 생성
    GameMessage gameMessage = new GameMessage();

    // 게임 시작 이벤트에 람다 식으로 메시지 출력 동작 등록
    eventManager.OnGameStart += () => gameMessage.ShowMessage("게임이 시작됩니다.");

    // 게임 종료 이벤트에 람다 식으로 메시지 출력 동작 등록
    eventManager.OnGameEnd += () => gameMessage.ShowMessage("게임이 종료됩니다.");

    // 게임 실행
    eventManager.RunGame();
}
```

03. Func과 Action

▼ 1) Func, Action 란?

- `Func`과 `Action`은 델리게이트를 대체하는 미리 정의된 제네릭 형식입니다.
- `Func`는 값을 반환하는 메서드를 나타내는 델리게이트입니다. 마지막 제네릭 형식 매개변수는 반환 타입을 나타냅니다. 예를 들어, `Func<int, string>`는 `int`를 입력으로 받아 `string`을 반환하는 메서드를 나타냅니다.
- `Action`은 값을 반환하지 않는 메서드를 나타내는 델리게이트입니다. `Action`은 매개변수를 받아들이지만, 반환 타입이 없습니다. 예를 들어, `Action<int, string>`은 `int`와 `string`을 입력으로 받고, 아무런 값을 반환하지 않는 메서드를 나타냅니다.
- `Func` 및 `Action`은 제네릭 형식으로 미리 정의되어 있어 매개변수와 반환 타입을 간결하게 표현할 수 있습니다.

▼ 2) 사용 예제

- **Func 예제**

```
// Func를 사용하여 두 개의 정수를 더하는 메서드
int Add(int x, int y)
{
    return x + y;
}

// Func를 이용한 메서드 호출
Func<int, int, int> addFunc = Add;
```

```
int result = addFunc(3, 5);
Console.WriteLine("결과: " + result);
```

- **Action 예제**

```
// Action을 사용하여 문자열을 출력하는 메서드
void PrintMessage(string message)
{
    Console.WriteLine(message);
}

// Action을 이용한 메서드 호출
Action<string> printAction = PrintMessage;
printAction("Hello, World!");
```

- **활용 예제**

```
// 게임 캐릭터 클래스
class GameCharacter
{
    private Action<float> healthChangedCallback;

    private float health;

    public float Health
    {
        get { return health; }
        set
        {
            health = value;
            healthChangedCallback?.Invoke(health);
        }
    }

    public void SetHealthChangedCallback(Action<float> callback)
    {
        healthChangedCallback = callback;
    }
}

// 게임 캐릭터 생성 및 상태 변경 감지
GameCharacter character = new GameCharacter();
character.SetHealthChangedCallback(health =>
{
    if (health <= 0)
    {
        Console.WriteLine("캐릭터 사망!");
    }
```

```
});

// 캐릭터의 체력 변경
character.Health = 0;
```

04. LINQ (Language Integrated Query)

▼ 1) LINQ란?

- .NET 프레임워크에서 제공되는 쿼리 언어 확장
- 데이터 소스(예: 컬렉션, 데이터베이스, XML 문서 등)에서 데이터를 쿼리하고 조작하는데 사용됩니다.
- 데이터베이스 쿼리와 유사한 방식으로 데이터를 필터링, 정렬, 그룹화, 조인 등 다양한 작업을 수행할 수 있습니다.
- LINQ는 객체, 데이터베이스, XML 문서 등 다양한 데이터 소스를 지원합니다.

▼ 2) 구조

```
var result = from 변수 in 데이터소스
             [where 조건식]
             [orderby 정렬식 [, 정렬식...]]
             [select 식];
```

- `var` 키워드는 결과 값의 자료형을 자동으로 추론합니다.
- `from` 절에서는 데이터 소스를 지정합니다.
- `where` 절은 선택적으로 사용하며, 조건식을 지정하여 데이터를 필터링합니다.
- `orderby` 절은 선택적으로 사용하며, 정렬 방식을 지정합니다.
- `select` 절은 선택적으로 사용하며, 조회할 데이터를 지정합니다.

▼ 3) 간단한 예제

```
// 데이터 소스 정의 (컬렉션)
List<int> numbers = new List<int> { 1, 2, 3, 4, 5 };

// 쿼리 작성 (선언적인 구문)
```

```
var evenNumbers = from num in numbers
                  where num % 2 == 0
                  select num;

// 쿼리 실행 및 결과 처리
foreach (var num in evenNumbers)
{
    Console.WriteLine(num);
}
```

이전 강의

📕 2) 예외 처리 및 값형과 참조형

다음 강의

14강 고급 자료형 및 기능