알림

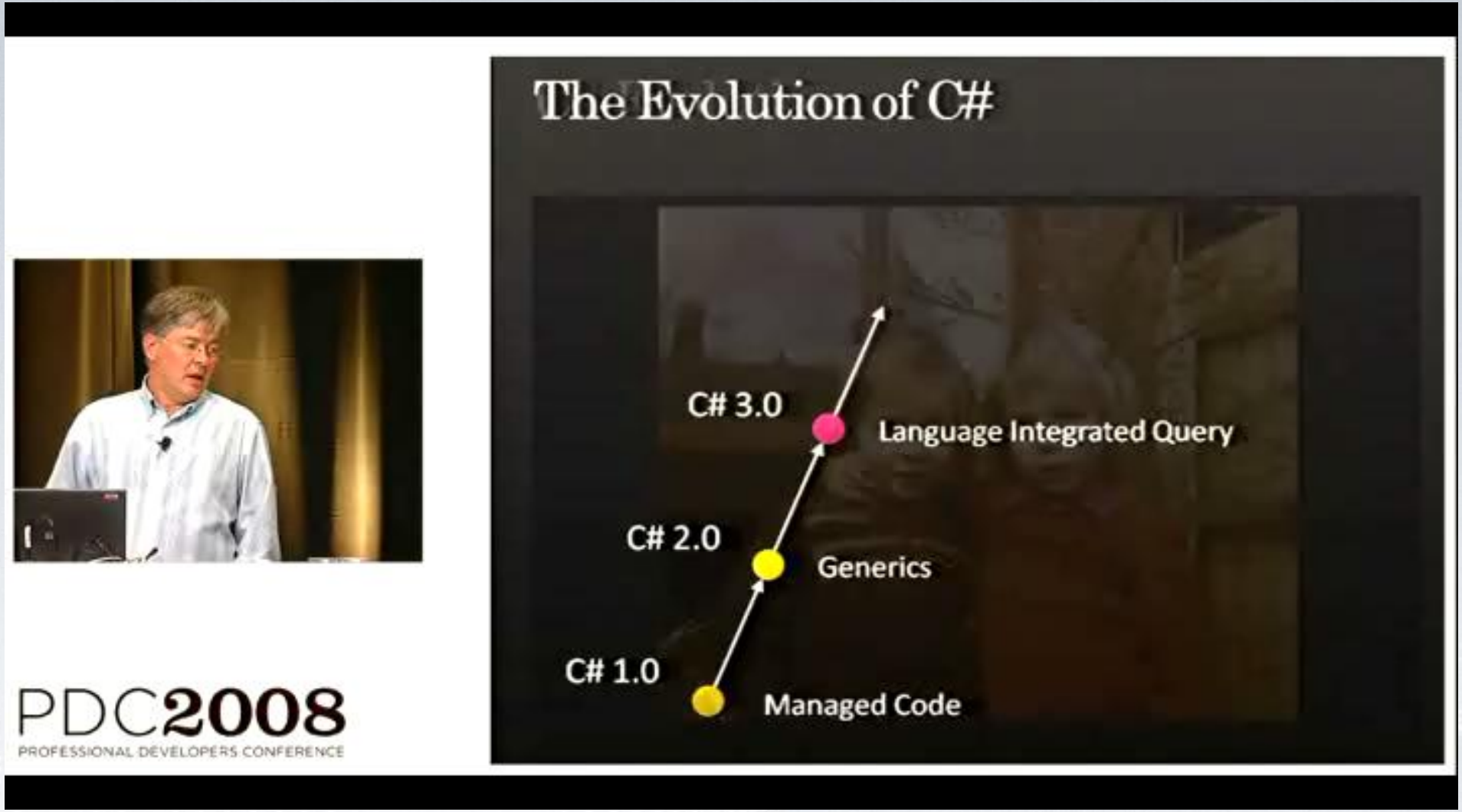
- 이 세션은 그 동안 진행했던 C# 4.0에 대한 세미나를 총정리한 마지막 세미나입니다. 처음 듣는 분도 잘 이해할 수 있도록 많은 노력을 했습니다. ☺
- 이 세션은 제가 6년 동안 '그것이 알고싶다'를 보면서 쌓은 노하우로 '그것이 알고싶다'와 최대한 비슷한 형태로 진행하려고 구성한 세션입니다. 재미를 위한 시도이니 오해없으시길 바랍니다 ☺
- 이 세션에서 인용된 영문의 번역은 제가 다 했으며, 정확도는 x%입니다.(단, x는 1부터 99사이의 정수) ☺

2010년 8월 28일

Visual Studio Camp #1

그것이 알고싶다 – C# 4.0의 변화, 그 진실은 무엇인가. 희망인가? 또 다른 혼란인가?

강보람
Visual C# MVP



앤더스 헬스버그 : 자, 이제 다시 **C#**의 진화에 대한 이야기로 돌아가보죠. 이제 부터는 **C#**의 다음 버전인 **C# 4.0**에 대해서 이야기 하겠습니다. 그리고 그 뒤에 우리가 작업중인 거에 대해서 좀 더 이야기 하죠. **C# 4.0**의 테마는 다이나믹 프로그래밍이라고 볼 수 있습니다.

What do you think of the new C# 4.0 'dynamic' keyword? - Stack Overflow - Windows Internet Explorer

http://stackoverflow.com/questions/244302/what-do-you-think-of-the-new-c-4-0-dynamic-keyword

Welcome to Q&A for professional and enthusiast programmers -- check out the FAQ!

link | flag

answered Oct 28 '08 at 18:48

Hugo

1,144 • 9 • 15

I'm sincerely disappointed by this new addition.

Compile-time type safety is a massively effective mechanism for eliminating a whole class of bugs.

Bugs that can't be detected until runtime can be very dangerous, particularly if they're associated with edge conditions.

While I acknowledge the need for dynamic typing in certain circumstances, I really think this should be an explicit opt-in design decision, either through the use of a compiler pragma or (preferably, IMO) from a different assembly written in a dynamically-typed language.

Factor in the performance overhead and there's very little positive about it for me.

난 이 새로운 기능에 완전히 실망했다.
컴파일타임의 타입안전성은 모든 종류의 버그를 효율적으로 제거할 수 있는 방법이다.
런타임까지 잡지 못한 버그는 위험한데, 특별히 경계 조건에 걸린 버그는 더 위험하다.
물론 다이나믹 타이핑이 유용한 상황에 대해서는 알고 있다. 하지만, 그런 상황은
컴파일러의 프라그마를 사용하거나 다른 동적 언어에서 만든 어셈블리를 사용하는 것 같이,
이미 서로 다이나믹 타이핑을 하기로 합의한 상황에서만 발생해야 한다.

be a better way :) – SoloBold Oct 29 '08 at 3:36

Oops... wrong thread... – SoloBold Oct 29 '08 at 3:38

show 2 more comments

My View of C# 4.0 - Omer van Kloeten's .NET Zen - Windows Internet Explorer

http://weblogs.asp.net/okloeten/archive/2008/10/29/6708812.aspx

Programming is life, the rest is mere details

HOME CONTACT RSS ATOM COMMENTS RSS

News

facebook

My View of C# 4.0

I've known a bit about C# 4.0 for a while now and have had time to think about it. I've just re-read the New features in C# 4.0 paper published by Microsoft and would like to offer the following critique of the language's new features:

Dynamic Lookup

This feature just makes me cringe, just like anonymous methods made me cringe when they were introduced in C# 2.0. To this day, I

The dynamic keyword is as open to abuse as anything could be. It takes the principles of static typing and throws the baby out with the bathwater.

Dynamic 키워드는 모든 경우에 악용될 소지가 있다. 이 키워드는 정적 타이핑의 모든 원칙을 마치 목욕물을 버리면서 아기까지 버리듯이 무시할 수 있다.

What is wrong with it

and the aspect of run time. Unfortunately, too much of a good thing is bad for you. Imagine the following:

```
public dynamic GetCustomer()
{
    // mystery...
}
```

What do we have here then? I don't know and neither does IntelliSense? I guess we'll have to go with trial and error. I admit this is quite the dramatization, but you get my point: it's ripe for abusing an otherwise perfectly fine static syntax.

Moreover, the dynamic keyword's syntax does what no other feature of C# has ever done - **it breaks existing syntax**. Should I define in C# 3.0 a type named dynamic, the following piece of code will take a whole different meaning in C# 4.0:

```
public dynamic GetCustomer()
{
    dynamic customer = GetCustomerCOMObject();
    return customer;
}
```

How it can be fixed

remember himself.

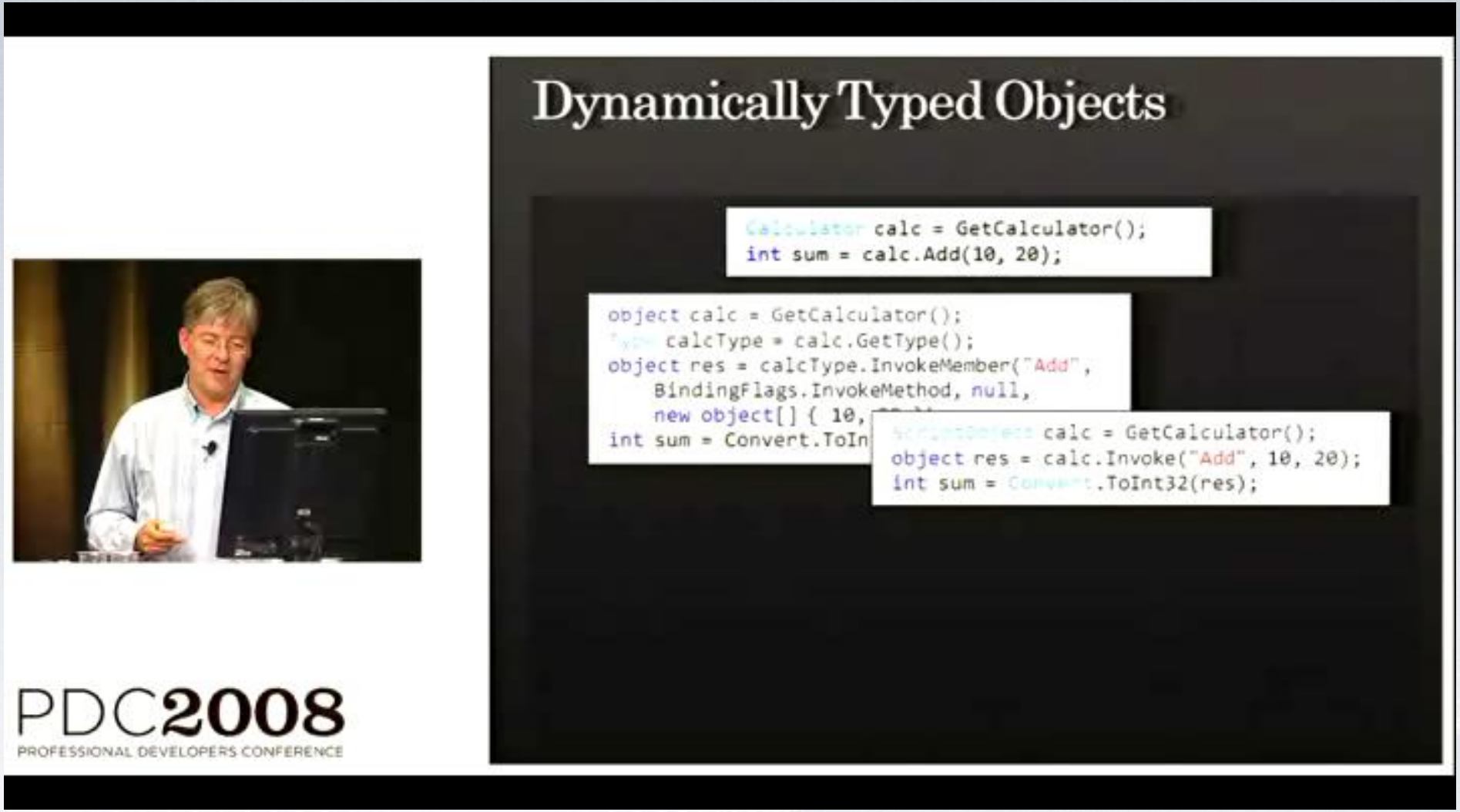
As a senior developer at NuConomy, a leading web analytics and advertising startup, he leads a wide range of technologies for its flagship products.

הבלוגים
blogs.microsoft.co.il

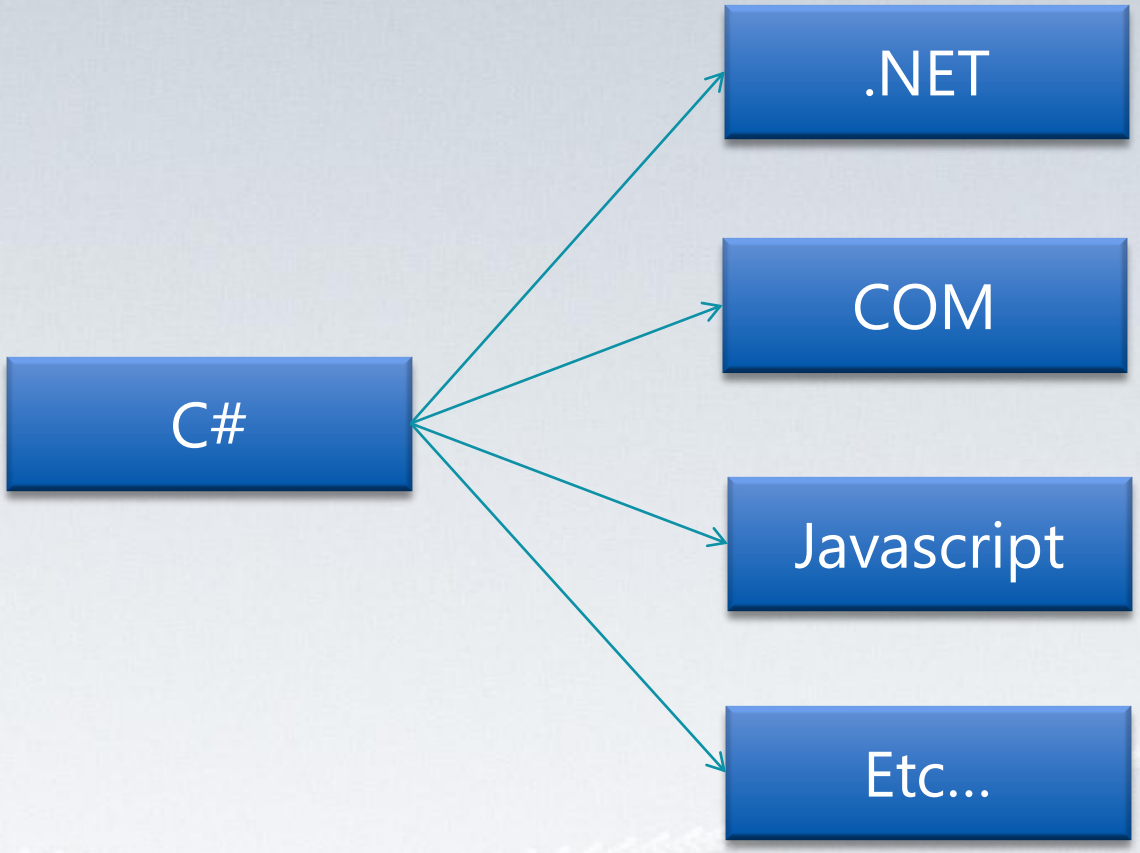
Take back the web

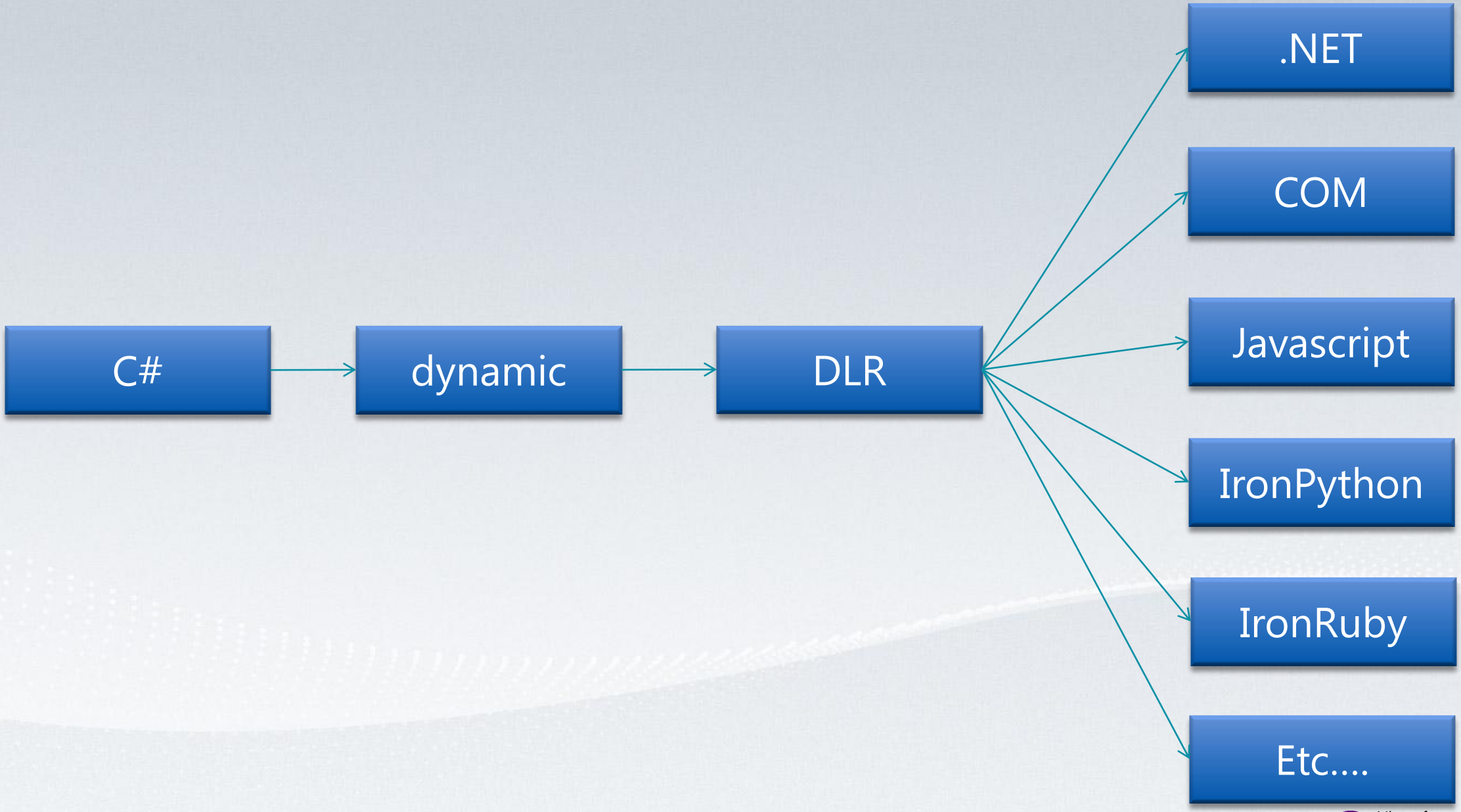
@omerk (twitter)

"The GoodGuide app helps you find



앤더스 헬스버그 : **4.0**에서는 코드를 이런식으로 작성할 수 있습니다. 간단하게 정적 타입이 **dynamic**인 변수를 하나 선언하는 거죠. **(웃음)**
즉, 정적으로 **dynamic**타입을 정해줄 수 있는 것입니다. 컴파일러 한테 이렇게 이야기하는 거죠, '이 변수가 뭔가의 참조를 가지고 있는데, 그 변수에 대해서 정적인 방식말고, 동적으로 연산을 수행하고 싶어'


C#
.NET
COM
Javascript
Etc…


C#
dynamic
DLR
.NET
COM
Javascript
IronPython
IronRuby
Etc....

dynamic 타입이란?

dynamic != object

dynamic >= object

object + Duck typing > = object

Duck typing이란?

```
public void 개들의모임에입장(Dog  dog)
{
    dog.멍멍멍();
}
```

```
Class Do
{
    pub void 멍멍() {}
}
```

```
Class 술먹고개가된사람
{
    public 멍멍() {}
}
```

```
public void 개들의모임에입장(dynamic  dog)
{
    dog.멍멍멍();
}
```

```
Class Dog
{
    public void 멍멍() {}
}
```

```
Class 술먹고개가된사람
{
    public void 멍멍() {}
}
```

왜 dynamic >= object 인가?

```
class Overload
{
    public static void DoNothing(object value)
    {
    }

    public static void DoNothing(dynamic value)
    {
    }

    static void Main(string[] args)
    {
    }
}
```

ConsoleApplication8 - Microsoft Visual Studio (관리자)

파일(F) 편집(E) 보기(V) 프로젝트(P) 빌드(B) 디버그(D) 팀(M) 데이터(A) 도구(T) 아키텍처(C) 테스트(S) 분석(N) 창(W) 도움말(H)

Debug | x86 | primitives:

Rules.py | Program.cs*

Chapter7.Overload | Main(string[] args)

```
1  using System;
2
3  namespace Chapter7
4  {
5      class Overload
6      {
7          public static void DoNothing(object value)
8          {
9          }
10
11         public static void DoNothing(dynamic value)
12         {
13         }
14
15         static void Main(string[] args)
16         {
17
18         }
19     }
20 }
21
```

오류 목록

	열	프로젝트
1 'Chapter7.Overload' 형식은 동일한 매개 변수 형식을 가진 'DoNothing' 멤버를 미리 정의합니다.	28	ConsoleApplication8

준비

```
public dynamic DynamicCall(dynamic d)
{
```

```
[return: Dynamic]
public object DynamicCall([Dynamic] object d)
{
```

그렇다면 dynamic처리는 어떻게?

```
dynamic fdc = new FirstDynamicClass();
fdc.Hello();
```

```
// Methods
private static void Main(string[] args)
{
    object fdc = new FirstDynamicClass();
    if (<Main>o__SiteContainer0.<>p__Site1 == null)
    {
        <Main>o__SiteContainer0.<>p__Site1 =
            CallSite<Action<CallSite, object>>
                .Create(Binder.InvokeMember(
                    CSharpBinderFlags.ResultDiscarded,
                    "Hello",
                    null,
                    typeof(HelloDynamic),
                    new CSharpArgumentInfo[]
                    {
                        CSharpArgumentInfo.Create(
                            CSharpArgumentInfoFlags.None,
                            null)
                    }));
    }
    <Main>o__SiteContainer0.<>p__Site1
        .Target(<Main>o__SiteContainer0.<>p__Site1, fdc);
}

// Nested Types
[CompilerGenerated]
private static class <Main>o__SiteContainer0
{
    // Fields
    public static CallSite<Action<CallSite, object>> <>p__Site1;
}
```

```
// Methods
private static void Main(string[] args)
{
    object fdc = new FirstDynamicClass();
    if (<Main>o__SiteContainer0.<>p__Site1 == null)
    {
        <Main>o__SiteContainer0.<>p__Site1 =
            CallSite<Action<CallSite, object>>
                .Create(Binder.InvokeMember( // 동적 콜 사이트를 생성.
                    CSharpBinderFlags.ResultDiscarded, // 리턴 타입이 void
                    "Hello", // 호출할 메서드의 이름
                    null, // 매개변수 없음
                    typeof(HelloDynamic), // 호출이 일어난 컨텍스트
                    new CSharpArgumentInfo[]  //호출을 위한 매개변수 값
                    {
                        CSharpArgumentInfo.Create(
                            CSharpArgumentInfoFlags.None,
                            null)
                    }));
    }
    <Main>o__SiteContainer0.<>p__Site1
        .Target(<Main>o__SiteContainer0.<>p__Site1, fdc);
}

// Nested Types
[CompilerGenerated]
private static class <Main>o__SiteContainer0
{
    // Fields
    public static CallSite<Action<CallSite, object>> <>p__Site1;
}
```

동적 콜 사이트(dynamic call site)란?

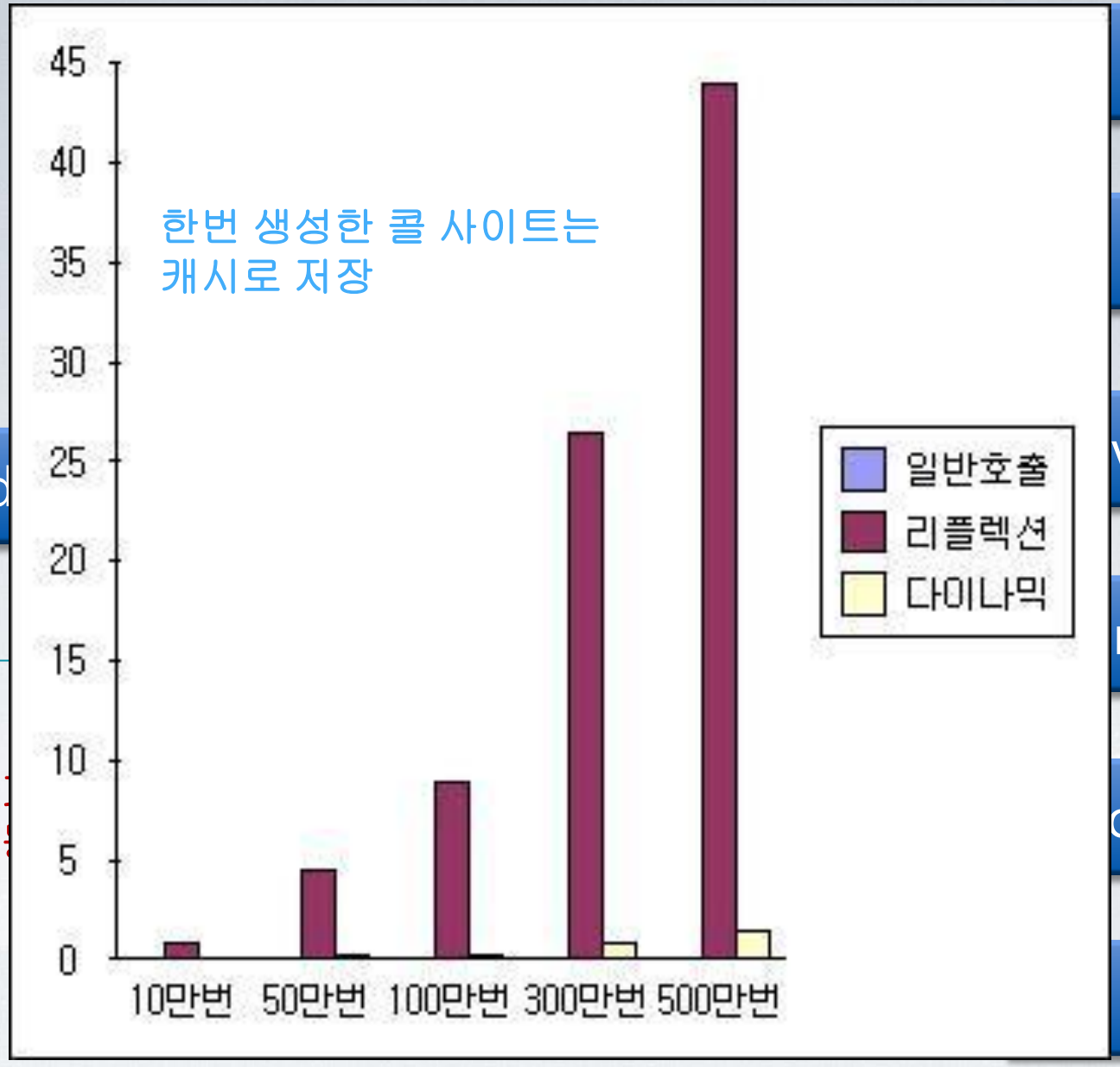
컴파일러가 소스를 분석하던 중, 동적 호출이 발생하는 곳에 동적 호출을 위해 필요한 작업을 처리해놓은 곳으로, 컴파일러가 DLR에게 '이곳에서 동적 호출이 발생했으니 런타임에 처리해달라'고 부탁하는 것이다.

C#

d

추

성



.NET

COM

vascript

nPython

onRuby

Etc....

Microsoft Visual Studio

사람들이 우려하던 상황
- 링 바이러스 시나리오

```
dynamic name = "세식이바보똥깨축구해삼멍게";
Console.WriteLine( name.ToString() );
```

dynamic타입인 **name**에 **ToString**메서드를 호출했으므로
동적 콜 사이트를 통해서 런타임에 바인딩 되며,
결과타입은 **dynamic**

WriteLine메서드에 **dynamic**타입의 매개변수가 넘어왔으므로
WriteLine의 어떤 오버로드에 바인딩 될지는 런타임에 결정된다.
그래서 **WriteLine**에 대한 호출도 동적 콜사이트를 통해 **DLR**이 처리한다.

이런식으로 **dynamic**타입을 남발하게 되면, **dynamic**이 **dynamic**을 낳는 상황이 되므로,
링 바이러스처럼 코드 전체에 **dynamic**이 퍼지게 된다.
그리고 심각한 성능의 저하 뿐만 아니라, 이해하기 힘든 코드가 된다.

```
public static void Runtime()
{
    dynamic sum = 0;

    for (dynamic i = 0; i < 100000; i++)
    {
        sum += i;
    }

    Console.WriteLine(sum.ToString());
}
```

```
public static void Runtime()
{
    object sum = 0;
    object i = 0;
Label_00BD:
    if (<Runtime>o__SiteContainer0.<>p__Site3 == null)
    {
        <Runtime>o__SiteContainer0.<>p__Site3 =
            CallSite<Func<CallSite, object, bool>>
            .Create(Binder.UnaryOperation(
                CSharpBinderFlags.None,
                ExpressionType.IsTrue,
                typeof(PerformanceTest),
                new CSharpArgumentInfo[]
                    { CSharpArgumentInfo.Create(CSharpArgumentInfoFlags.None, null) }));
    }
    if (<Runtime>o__SiteContainer0.<>p__Site4 == null)
    {
        <Runtime>o__SiteContainer0.<>p__Site4 =
            CallSite<Func<CallSite, object, int, object>>
            .Create(Binder.BinaryOperation(
                CSharpBinderFlags.None,
                ExpressionType.LessThan,
                typeof(PerformanceTest),
                new CSharpArgumentInfo[] {
                    CSharpArgumentInfo.Create(CSharpArgumentInfoFlags.None, null),
                    CSharpArgumentInfo.Create(CSharpArgumentInfoFlags.Constant | CSharpArgumentInfoFlags.UseCompileTimeType, null) }));
    }
    if (<Runtime>o__SiteContainer0.<>p__Site3
        .Target(<Runtime>o__SiteContainer0.<>p__Site3,
            <Runtime>o__SiteContainer0.<>p__Site4
                .Target(<Runtime>o__SiteContainer0.<>p__Site4, i, 0x186a0)))
    {
        if (<Runtime>o__SiteContainer0.<>p__Site1 == null)
        {
            <Runtime>o__SiteContainer0.<>p__Site1 =
                CallSite<Func<CallSite, object, object, object>>
                .Create(Binder.BinaryOperation(
                    CSharpBinderFlags.None,
                    ExpressionType.AddAssign,
                    typeof(PerformanceTest),
                    new CSharpArgumentInfo[] {
                        CSharpArgumentInfo.Create(CSharpArgumentInfoFlags.None, null),
                        CSharpArgumentInfo.Create(CSharpArgumentInfoFlags.None, null) }));
        }
        sum = <Runtime>o__SiteContainer0.<>p__Site1
            .Target(<Runtime>o__SiteContainer0.<>p__Site1, sum, i);

        object CS$0$0001 = i;
        if (<Runtime>o__SiteContainer0.<>p__Site2 == null)
        {
            <Runtime>o__SiteContainer0.<>p__Site2 =
                CallSite<Func<CallSite, object, object>>
                .Create(Binder.UnaryOperation(
                    CSharpBinderFlags.None,
                    ExpressionType.Increment,
                    typeof(PerformanceTest),
                    new CSharpArgumentInfo[] {
                        CSharpArgumentInfo.Create(CSharpArgumentInfoFlags.None, null) }));
        }
        i = <Runtime>o__SiteContainer0.<>p__Site2
            .Target(<Runtime>o__SiteContainer0.<>p__Site2, CS$0$0001);
        goto Label_00BD;
    }
    if (<Runtime>o__SiteContainer0.<>p__Site5 == null)
    {
        <Runtime>o__SiteContainer0.<>p__Site5 = CallSite<Action<CallSite, Type, object>>
            .Create(Binder.InvokeMember(
                CSharpBinderFlags.ResultDiscarded,
                "WriteLine",
                null,
                typeof(PerformanceTest),
                new CSharpArgumentInfo[] {
                    CSharpArgumentInfo.Create(CSharpArgumentInfoFlags.IsStaticType | CSharpArgumentInfoFlags.UseCompileTimeType, null),
                    CSharpArgumentInfo.Create(CSharpArgumentInfoFlags.None, null) }));
    }
    if (<Runtime>o__SiteContainer0.<>p__Site6 == null)
    {
        <Runtime>o__SiteContainer0.<>p__Site6 = CallSite<Func<CallSite, object, object>>
            .Create(Binder.InvokeMember(
                CSharpBinderFlags.None,
                "ToString",
                null,
                typeof(PerformanceTest),
                new CSharpArgumentInfo[] {
                    CSharpArgumentInfo.Create(CSharpArgumentInfoFlags.None, null) }));
    }
    <Runtime>o__SiteContainer0.<>p__Site5
        .Target(<Runtime>o__SiteContainer0.<>p__Site5, typeof(Console),
            <Runtime>o__SiteContainer0.<>p__Site6.
                Target(<Runtime>o__SiteContainer0.<>p__Site6, sum));
}
```

링 바이러스에 대처하는 우리의 자세
-우물에 짱 박아라

박세식(29) / 링 바이러스 퇴치 전문가

그럼요, 우물에 짱 박는게 최고죠. 제가 그렇게 40년 동안 고놈들 잡아 왔거든요. 이것들이 처음에는 기세 당당하게 입 쩍~벌리고 이상한 소리내면서 나올라고 버티는데요, 한 몇일 지나서 물 다 떨어지고 하면, 살려달라고 울고 난리도 아니죠. 네? 제 나이랑 경력이랑 안 맞다구요? 니가 뭔데 나이를 묻고 그래? 응?

1. Dynamic은 메서드 안에서 격리되도록 한다.
- 메서드안에서 dynamic이 격리되어서 확산되지 않도록 한다.

2. 절대 리턴 타입에 dynamic을 쓰지 말 것.
- 리턴 타입이 어떤 타입인지 알 수 있는 방법이 없을 뿐 더러, 링 바이러스를 격리시킬 수 없다.

3. 성능이 중요한 곳에서는 절대로 쓰지 말 것.
- 아무리 동적 콜 사이트에서 캐시를 사용한다고 하더라도, 성능이 떨어지므로 성능이 중요하다면, 사용하지 말자.

그런데 왜 동적 언어의 개념을 추가하는가?

박세식(29) / 아까 경력으로 구라친 넘

그런데, 왜 이런걸 추가해서 이 난리를 치는겁니까? 예?
안 그래도 새로운 거 나오면 배우느라고 힘들어 죽겠는데.
당신들 계속 이러면 정말 재미 없어.

매년 새로운 언어를 최소 하나는 배워라. **다른 언어는 동일한 문제를 다르게 풀기 때문**이다.
- '실용주의 프로그래머'

생각, 느낌 따위를 나타내거나 전달하는 데에 쓰는 음성, 문자 따위의 수단.
또는 그 음성이나 문자 따위의 **사회 관습적인 체계**
- 네이버 국어사전

D사의 한 임원은 "상사와 회의를 할 때 영어로 하면 내 의견을 끝까지 다 듣고 합리적으로 대화를 하는데, 우리나라 말로 하면 상당히 권위적으로 나오는 경향이 있다.
그래서 가급적 영어로 대화를 한다"며 **언어에 따라 커뮤니케이션 패턴이 달라질 수 있음**을 설명했다.
- LGERI 리포트 "조직 내 침묵 현상", 황인경, 박지원 , 2008. 12. 3

새로운 언어를 배우지 않아도, 기존에 쓰는 C#에서 동적인 언어의 특징을 **선택적으로** 활용할 수 있게 되었다.

1. C#은 현대적인 언어를 지향하므로,
   앞으로도 가장 현대적인 기능들을 추가해 나갈 것이다.

2. C# 4.0의 변화는 동적 언어의 사용을 권장하는 것이 아니다.
   - C#은 앞으로도 계속해서 정적 언어로 남을 것이다.

3. 정적 언어의 한계로 불가능하거나,
   불편한 부분을 동적 언어를 통해서 보완하라는 것이다.

4. 그리고 닷넷을 타겟으로 Ruby나 Python같은
   동적 언어를 활용할 수 있는 길이 열렸다.

동적 언어의 테크닉을 활용하는 예

C# 4.0은 dynamic이 끝인가?

Covariance와 Contravariance

입력된 정수를 두 배 해서 정수위로 투영하는 투영 D가 있다. 즉, D(x) = 2x이다.

x = -1, y = 2 일 경우

-1 <= 2 이므로
x <= y 만족

D(x) = 2 * x

D(x) = D(-1) = -2

D(y) = D(2) = 4

-2 <= 4 이므로
D(x) <= D(y) 만족

항상 x <= y 이면,
D(x) <= D(y)이므로
공변성을 가진다.
(Covariant하다)

입력된 정수의 부호를 반대로 해서 정수 위로 투영하는 투영 N이 있다.
즉, N(x) = -1 * x이다.

x = -1, y = 2 일 경우

-1 <= 2 이므로
x <= y 만족

N(x) = -1 * x

N(x) = N(-1) = 1

N(y) = N(2) = -2

1 >= -2 이므로
N(x) >= N(y)가 되므로
원래의 관계가 역전됨

항상 x <= y 이면,
N(x) >= N(y)이므로
반공변성을 가진다.
(Contravariant하다)

Covariance의 안 좋은 예

```
string[] sa = new string[2];
object[] oa = sa;
oa[0] = "hello";
oa[1] = 5;
string s = (string)oa[1];
Console.WriteLine(s);
```

처리되지 않은 예외: System.ArrayTypeMismatchException: 배열과 호환되지 않는 형식으로 요소를 액세스하려고 했습니다.

```
public interface IEnumerable<out T> : IEnumerable
{
    IEnumerator<T> GetEnumerator();
}
```

IEnumerator<T>를 리턴할 뿐.

```
public interface IEnumerator<out T> : IDisposable, IEnumerator
{
    T Current { get; }
}
```

T타입의 객체를 리턴할 뿐.

모두 읽기 전용이므로, 배열과 같이 엉뚱한 값이 들어갈 위험성이 없다.

즉, out으로 선언된 타입 T는 매개변수의 자리에 들어가서는 안된다.
그리고 out으로 선언된 T에 대해서는 공변성을 통한 형변환이 지원된다.

```
public static void DoSth(IEnumerable<object> list)
{
    //뭔가 하는 거임.
}

static void Main(string[] args)
{
    List<string> stringList = new List<string>();
    DoSth(stringList);
}
```

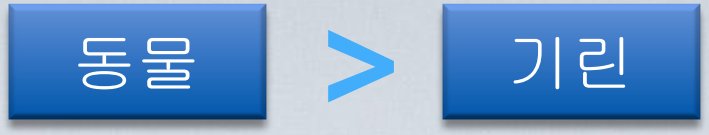
Ienumerable<T>는 공변성을 지원하므로, 문제없이 실행된다.

```
public interface IComparer<in T>
{
    int Compare(T x, T y);
}
```

타입 T를 입력받기만 할 뿐,
T를 다시 리턴하지 않는다.

T에 대해서는 입력만 가능한 형태이다.

즉, T는 리턴할 타입으로 사용되어서는 안된다.
그리고 in으로 선언된 T에 대해서는 반공변성을 통한 형변환이 지원된다.



동물 기린1 = new 기린();

기린 기린1 = new 동물();

```
IComparer<동물> 동물비교 = new 동물Comparer();
IComparer<기린> 기린비교 = 동물비교;

기린 기린1 = new 기린 { 이름 =
"니가그린기린그림은", 목길이 = 10 };
기린 기린1 = new 기린 { 이름 =
"내가그린기린그림은", 목길이 = 15 };

int result1 = 기린비교.Compare(기린1, 기린2);
```

동물 > 기린의 방향성을 뒤집어서 **IComparer<기린> > Icomparer<동물>**의 관계가 되었다.

간단하게 생각해서,

1. 동물을 비교할 수 있다면, 기린을 비교할 수 있을 것이다.

2. 하지만 기린을 비교할 수 있다고 해서, 모든 동물을 비교할 수 있는 것은 아니다.

3. 그래서 IComparer<T>의 경우는, 반공변성을 지원하는 것이다.

Named and Optional Parameters

```
static void CalcRatio(
    double fact1 = 90.0,
    double fact2 = 0.9887,
    double fact3 = 33.211)
{
    Console.WriteLine(fact1 * fact2 * fact3);
}

static void Main(string[] args)
{
    CalcRatio(); // CalcRatio(90.0, 0.9887, 33.211);
    CalcRatio(22.1, 0.8891); // CalcRatio(22.1, 0.8891, 33.211);
    CalcRatio(fact3: 40.12, fact1: 13.11); // CalcRatio(13.11, 0.9887, 40.12);
}
```

기본값을 설정해주는 것이
Optional parameter(선택적 파라미터)

어떤 파라미터에 어떤 매개변수를 넘겨줄지 지정하는 것이
Named argument(지명 매개변수)

COM Interop

PDC2008
PROFESSIONAL DEVELOPERS CONFERENCE

Improved COM Interoperability

```
object fileName = "Test.docx";
object missing  = System.Reflection.Missing.Value;

doc.SaveAs(ref fileName,
    ref missing, ref missing, ref missing,
    ref missing, ref missing, ref missing,
    ref missing, ref missing, ref missing,
    ref missing, ref missing, ref missing,
    ref missing, ref missing, ref missing);
```

앤더스 헬스버그 : 음, 이제 이렇게 복잡하게 하지 않아도 됩니다.
이제 **C# 4.0**에서는 짜짠~. 바로 **doc.SaveAs**메서드를 호출할 수 있죠.

C# 4.0의 테마는 크게 '다른 것들 과의 상호운용성 향상'이라고 볼 수 있다.

dynamic 타입과 DLR의 추가로 동적 언어와 상호운용이 가능해졌다.

Named and Optional parameters의 추가와 기타 COM관련 기능의 추가로
편의성과 함께 COM과의 상호운용이 편리해졌다.

Covariance와 Contravariance의 추가로
다른 타입을 담은 컬렉션 간의 형변환이 쉬워졌다.

참고자료

PDC 2008 - The Future of C#

http://www.vsts2010.net/category/C%23

http://blogs.msdn.com/ericlippert/archive/2009/11/30/what-s-the-difference-between-covariance-and-assignment-compatibility.aspx

http://stackoverflow.com/questions/244302/what-do-you-think-of-the-new-c-4-0-dynamic-keyword

http://weblogs.asp.net/okloeten/archive/2008/10/29/6708812.aspx

그 동안 봤던 수많은 '그것이 알고싶다'

쌩유☺

한국 Visual Studio 공식 팀
http://vsts2010.net
@vsts2010