Microsoft

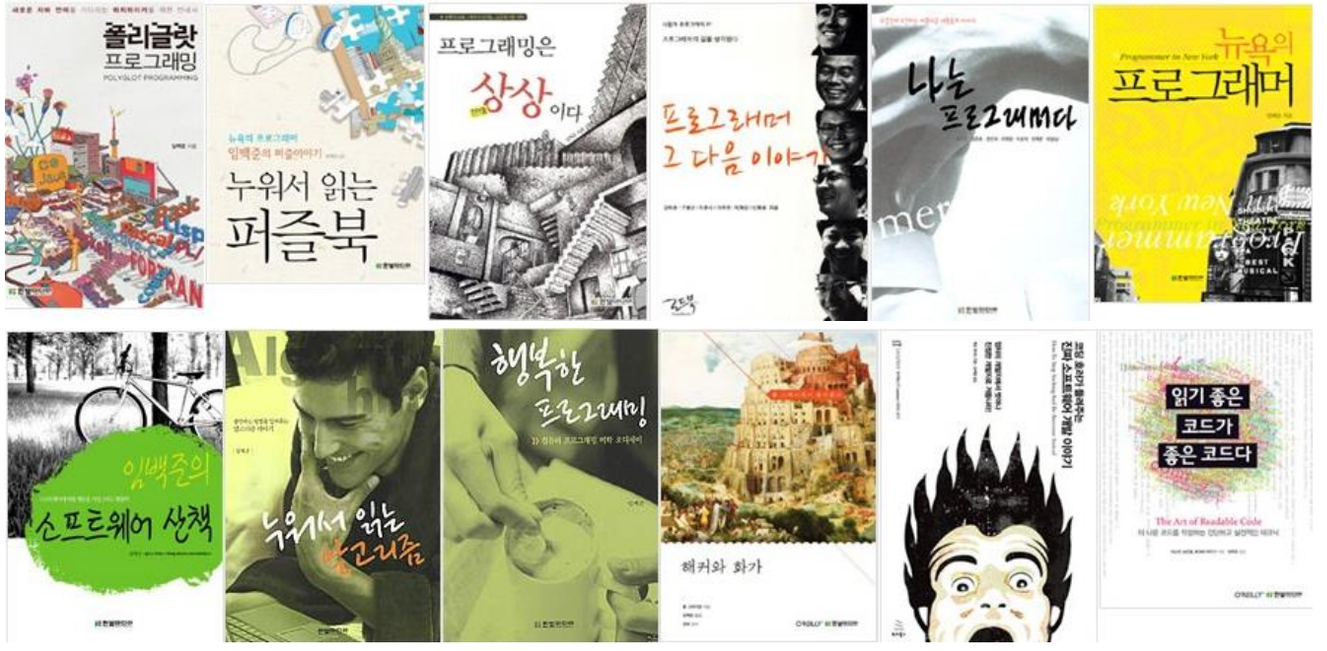
tech·days Korea 2014
개발자 컨퍼런스

자바개발자를 위한 C#

임백준

강사소개

tech·days Korea 2014
개발자 컨퍼런스



임백준

프로그래머 in 뉴욕

목차

C#의 역사

- C# 1.0 (2002/01) – 속성, 대리자, 값 타입
- C# 2.0 (2005/11) – 제네릭
- C# 3.0 (2007/11) – 링큐, 람다, 확장 메서드, 표현 트리
- C# 4.0 (2010/04) – 동적 바인딩, 제네릭 공변/반공변
- C# 5.0 (2012/08) – 비동기 메서드
- C# 6.0 (?) - 로슬린

제네릭

- Erasure vs Reification
- 자바 지우개 제네릭
- C# 구체화 제네릭
- 자바 지우개 제네릭의 한계
  - 타입조사 (introspection)
  - 오버로드 (overload)
  - 객체생성 (instantiation)
  - 리플랙션 (reflection)
  - 성능 (performance)

제네릭 – 타입조사


```java
public static <E> void add(E fruit) {

    if (fruit instanceof Apple) {
        System.out.println("I got an apple.");

    } else if (fruit instanceof List<?>) {
        System.out.println("I got an apple basket.");
    }
}
```

```csharp
public static void Add<E>(E fruit)
{
    if (fruit.GetType() == typeof(Apple))
    {
        Console.Out.WriteLine("I got an apple.");
    }
    else if (fruit.GetType() == typeof(List<Apple>))
    {
        Console.Out.WriteLine("I got an apple basket.");
    }
}
```

제네릭 – 오버로드


```
public class Overload<S, T> {

    public void foo(T t) {

    }

    public void foo(S s) {

    }

}
```

```
public void foo(S s)
{
    Console.Out.WriteLine(">>> I got S...");
}

public void foo(T t)
{
    Console.Out.WriteLine(">>> I got T...");
}
```

제네릭 – 객체생성


```
public class Instantiation {

    public <T> void foo()  {

        T t = new T();

    }

}
```

```
class Instantiation
{
    public void Foo<T>() where T : new() {
        T t = new T();
    }

    public static void Run()
    {
        Instantiation test = new Instantiation();
        test.Foo<int>();
        test.Foo<Dictionary<int, int>>();

    }
}
```

리플랙션


```
class GenericReflection
{
    public void Foo<T>(T t)
    {
        Console.Out.WriteLine(">>> T's type is " + t.GetType());
    }

    public void DoReflection()
    {
        object[] parameters = new string[] { "hello" };
        Type myType = typeof(GenericReflection);

        MethodInfo fooMethod = myType.GetMethod("Foo");
        MethodInfo generic = fooMethod.MakeGenericMethod(typeof(string));
        generic.Invoke(this, parameters);

    }

    public static void Run()
    {
        GenericReflection r = new GenericReflection();
        r.DoReflection();
    }
}
```

연산자 오버로드

- 제임스 고슬링 – 단순함이라는 자바의 철학
- 앤더스 하일스버그 – C++의 철학을 최대한 계승
- 가이 스티일 – 언어의 성장 (Growing a Language)

예외

- 실행시간 예외 (Runtime exception)
  - NullPointerException
  - IndexOutOfBoundsException
- 검사된 예외 (Runtime exception)
  - 매서드 시그너처의 일부
  - 콜사이트에서 반드시 try-catch 구문을 사용해야 함
- 앤더스 하일스버그 vs 제임스 고슬링

타입시스템

- 자바
  - 참조 타입 (Reference type)
  - 원시 타입 (Primitive type)
- C#
  - 참조 타입 (Reference type)
  - 값 타입 (Value type)
- 원시 타입 vs 값 타입

링큐

- 언어와 통합된 질의 (Language Integrated Query)
  - Linq to Object, Linq to XML, Ling to SQL
- 링큐를 가능하게 만든 C#의 문법
  - 질의 표현 (Query expression)
  - 암묵적 타입 변수 (Implicitly typed variables)
  - 객체와 컬랙션 초기자 (Object and collection initializers)
  - 익명 타입 (Anonymous types)
  - 확장 메서드 (Extension methods)
  - 람다 표현 (Lambda expressions)
  - 자동으로 구현되는 속성 (Auto-implemented properties)
  - 구체화 제네릭 (Reification)

링큐 – 질의 표현


```
class QueryExpressionTest
{
    public static void Run()
    {
        var person = from p in Data.GetPeople()
                     where p.Age > 12
                     select new { Age = p.Age, Name = "ABC" };


        var person2 = Data.GetPeople().
            Where(p => p.Age > 12).
            Select(p => new { Age = p.Age, Name = "ABC" });

        Console.Out.WriteLine("new person = " + person.First().Name);

        Console.Out.WriteLine("new person = " + person2.First().Name);

    }


}
```

링큐 – 확장 메서드


```
class ExtensionMethodTest
{
    public static void Run()
    {
        List<Person> people = Data.GetPeople();

        foreach (Person p in people)
        {
            Console.Out.WriteLine("Last name = " + p.GetLastName());
        }
    }
}

static class EM
{
    public static string GetLastName(this Person p)
    {
        return p.Name.Split(' ')[0];
    }
}
```

람다

- 자바 8
  - 람다
  - 함수 인터페이스 (Functional interface)
  - 메서드 참조 (Method reference)
  - 스트림
  - 디폴트 메서드
- C# 3.0
  - 람다
  - 대리자 (Delegates)
  - 링큐
  - 확장 메서드

람다 – 인터페이스 오염


```java
public static void main(String[] args) {

    IntToDoubleFunction f = n -> n / 2;

    System.out.println("half = " + half(f, 100));
}

public static double half(IntToDoubleFunction f, int n) {

    return f.applyAsDouble(n);

}
```

```
public static void Run()
{
    Func<int, double> f = n => n / 2;

    Console.Out.WriteLine("half = " + Half(f, 100));

}

public static double Half(Func<int, double> f, int n)
{
    return f(n);
}
```

Thank you