Generics in .NET

Joe Wirtley

@JoeWirtley

Wirtley Consulting LLC

About Me

Wirtley Consulting LLC

Springboro, OH

Dayton .NET Developer Group

C#, WPF, MVC, Web API

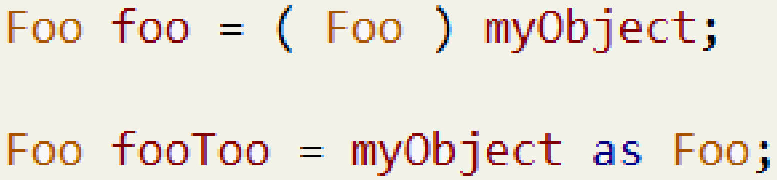
@JoeWirtley

Outline

- Basics
- Constraints
- Generics in action
- Other topics

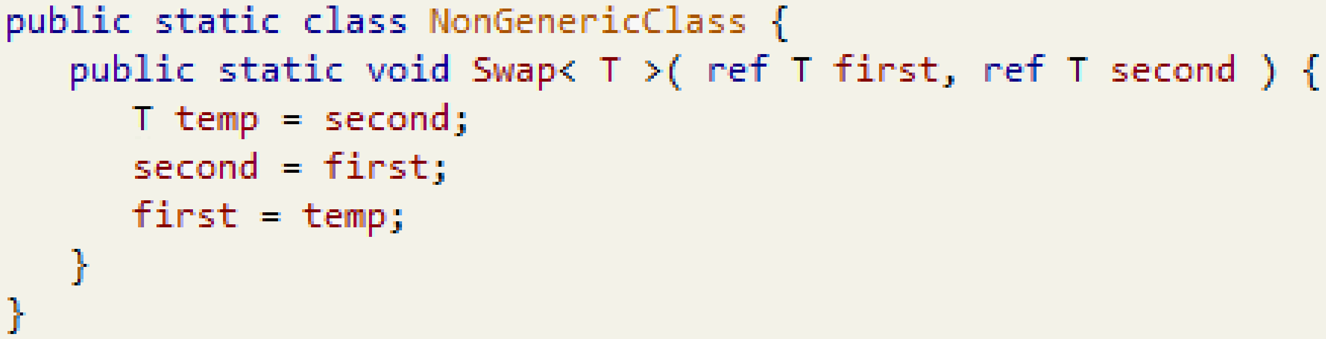
When to Use

- Code the same except for type
- When you are typecasting



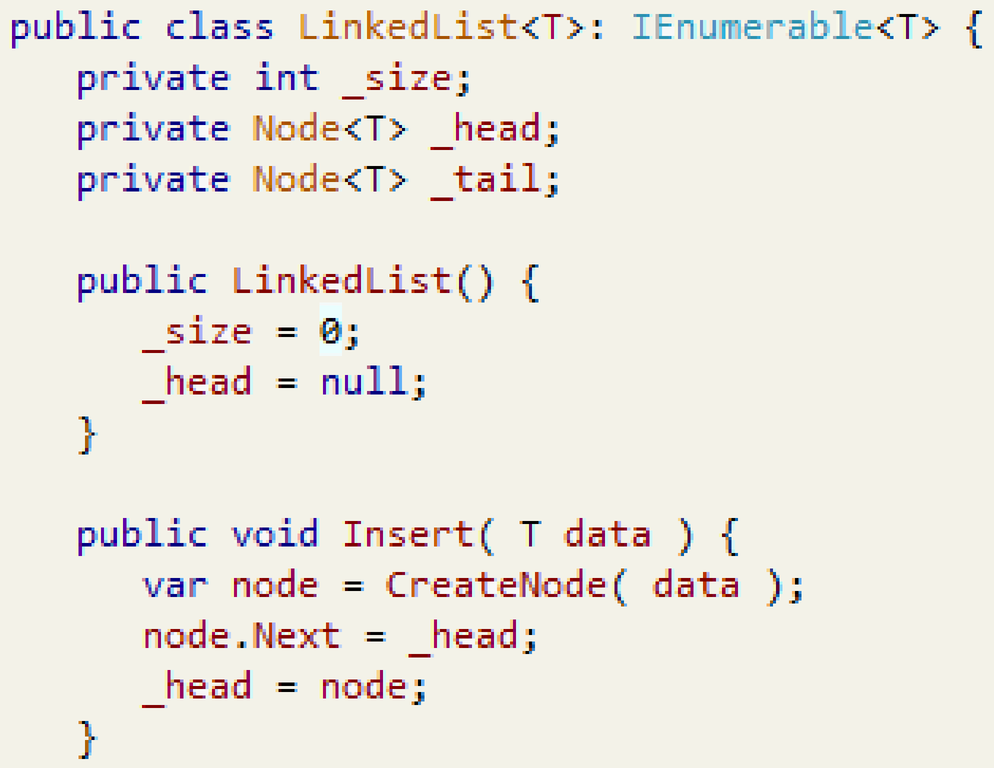
```
Foo foo = ( Foo ) myObject;

Foo fooToo = myObject as Foo;
```

Method



```
public static class NonGenericClass {
   public static void Swap< T >( ref T first, ref T second ) {
      T temp = second;
      second = first;
      first = temp;
   }
}
```

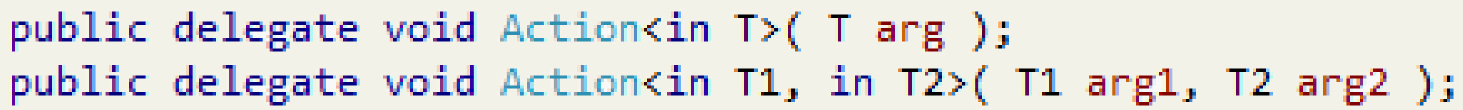
Classes and Interfaces



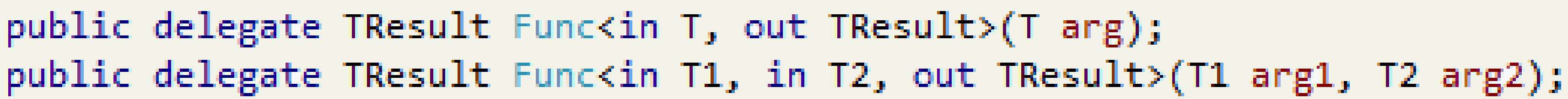
```
public class LinkedList<T>: IEnumerable<T> {
   private int _size;
   private Node<T> _head;
   private Node<T> _tail;

   public LinkedList() {
      _size = 0;
      _head = null;
   }

   public void Insert( T data ) {
      var node = CreateNode( data );
      node.Next = _head;
      _head = node;
   }
```

Wirtley
Consulting LLC

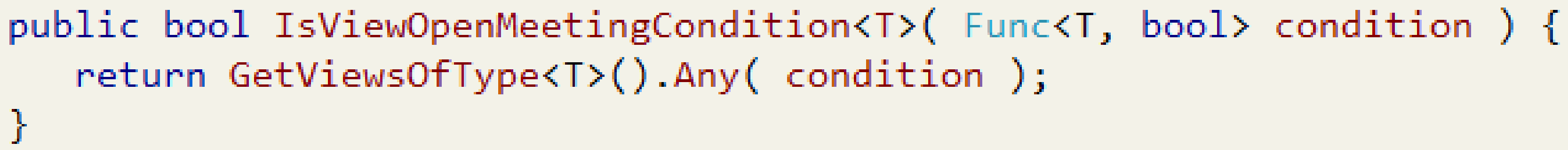
Delegates



```
public delegate void Action<in T>( T arg );
public delegate void Action<in T1, in T2>( T1 arg1, T2 arg2 );
```



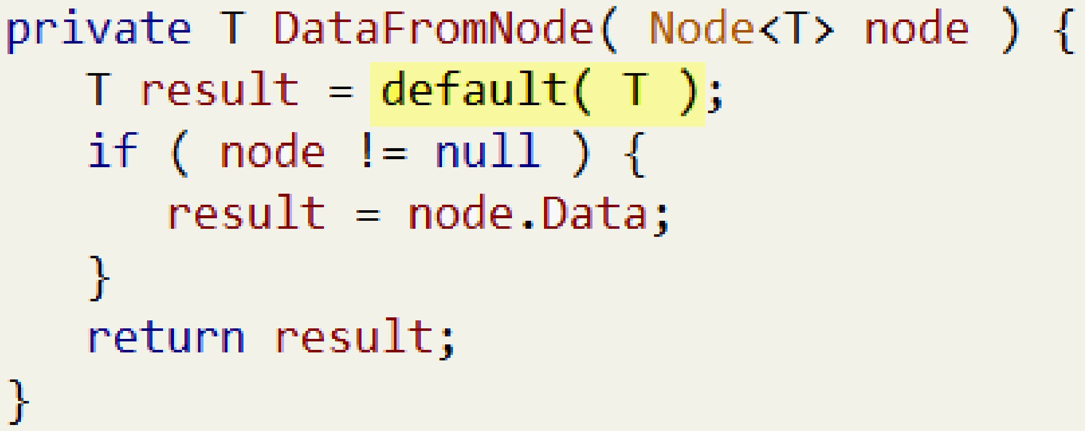
```
public delegate TResult Func<in T, out TResult>(T arg);
public delegate TResult Func<in T1, in T2, out TResult>(T1 arg1, T2 arg2);
```



```
public bool IsViewOpenMeetingCondition<T>( Func<T, bool> condition ) {
   return GetViewsOfType<T>().Any( condition );
}
```

Wirtley
Consulting LLC

Default



```
private T DataFromNode( Node<T> node ) {
   T result = default( T );
   if ( node != null ) {
      result = node.Data;
   }
   return result;
}
```

Basic Example

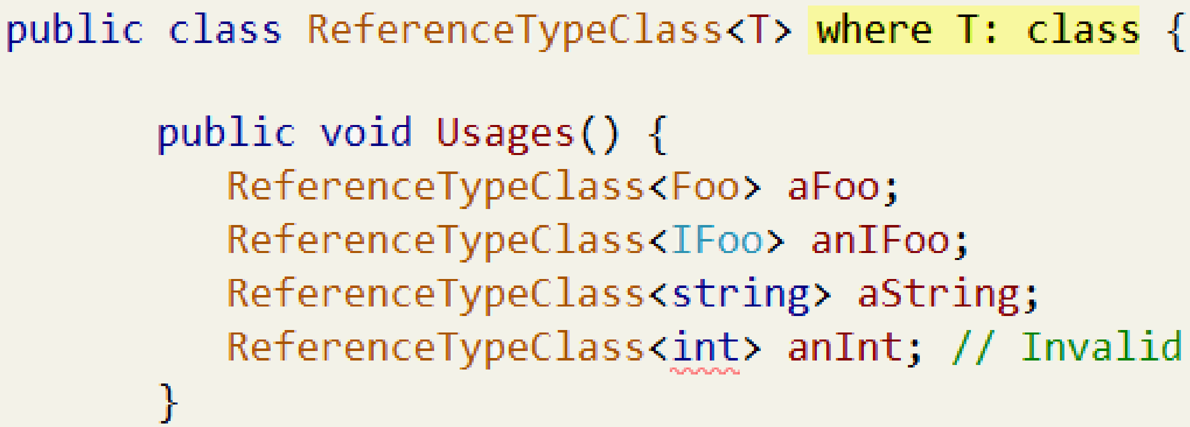
Constraints

- Restricts types that can be used in a generic

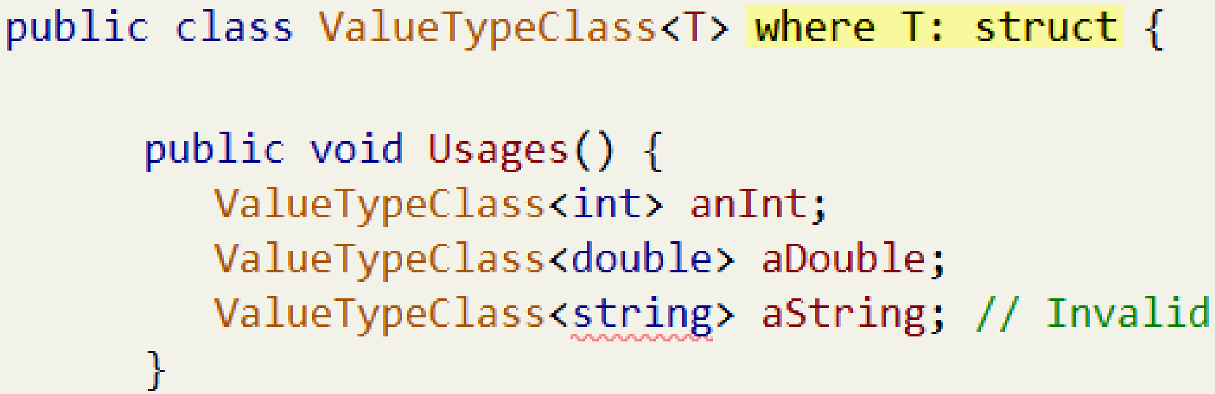
- On class or method

- Three types
  - Reference/Value
  - Type
  - New

Reference/Value Constraints

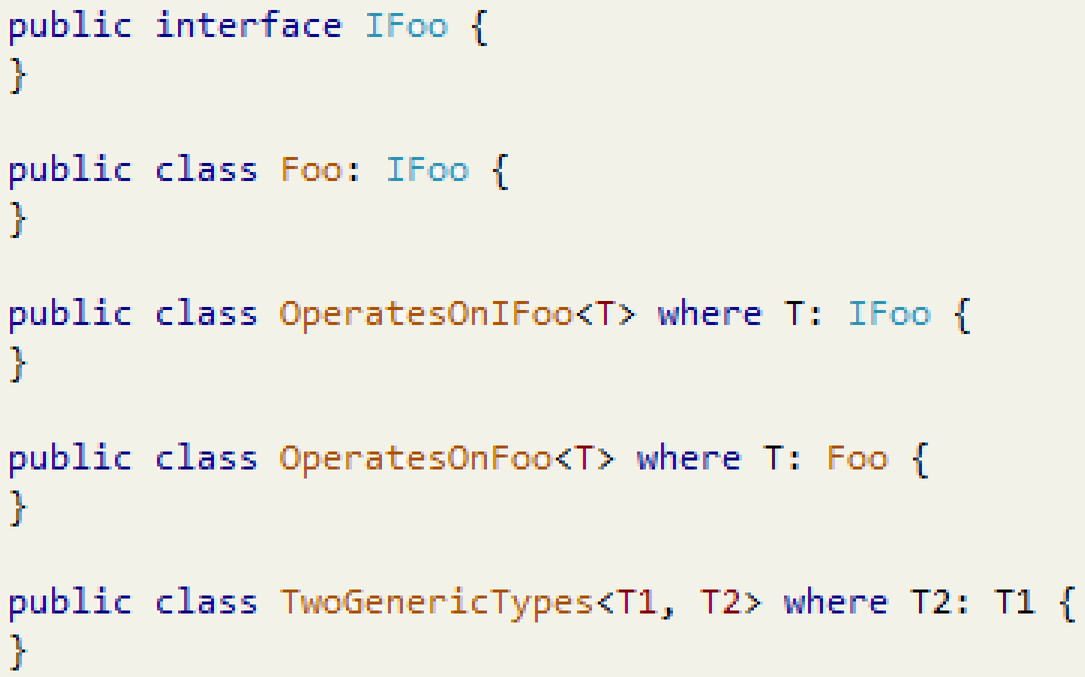


```
public class ReferenceTypeClass<T> where T: class {

      public void Usages() {
         ReferenceTypeClass<Foo> aFoo;
         ReferenceTypeClass<IFoo> anIFoo;
         ReferenceTypeClass<string> aString;
         ReferenceTypeClass<int> anInt; // Invalid
      }
```



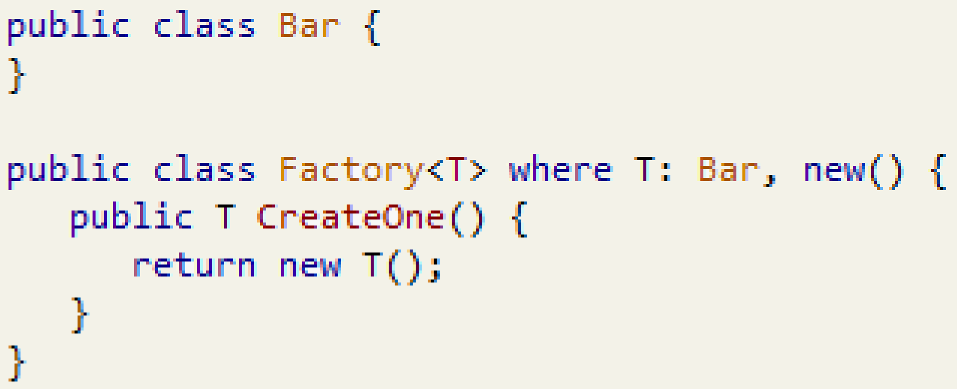
```
public class ValueTypeClass<T> where T: struct {

      public void Usages() {
         ValueTypeClass<int> anInt;
         ValueTypeClass<double> aDouble;
         ValueTypeClass<string> aString; // Invalid
      }
```

Type Constraints



```
public interface IFoo {
}

public class Foo: IFoo {
}

public class OperatesOnIFoo<T> where T: IFoo {
}

public class OperatesOnFoo<T> where T: Foo {
}

public class TwoGenericTypes<T1, T2> where T2: T1 {
}
```

New Constraint



```
public class Bar {
}

public class Factory<T> where T: Bar, new() {
    public T CreateOne() {
        return new T();
    }
}
```

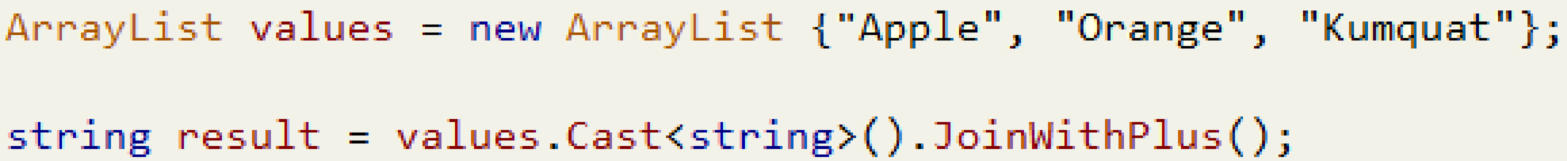
Unconstrained Type Parameters

- Cannot use == and != operators
- Can be converted to and from classes or interfaces
- Can compare to null

When constrained to class

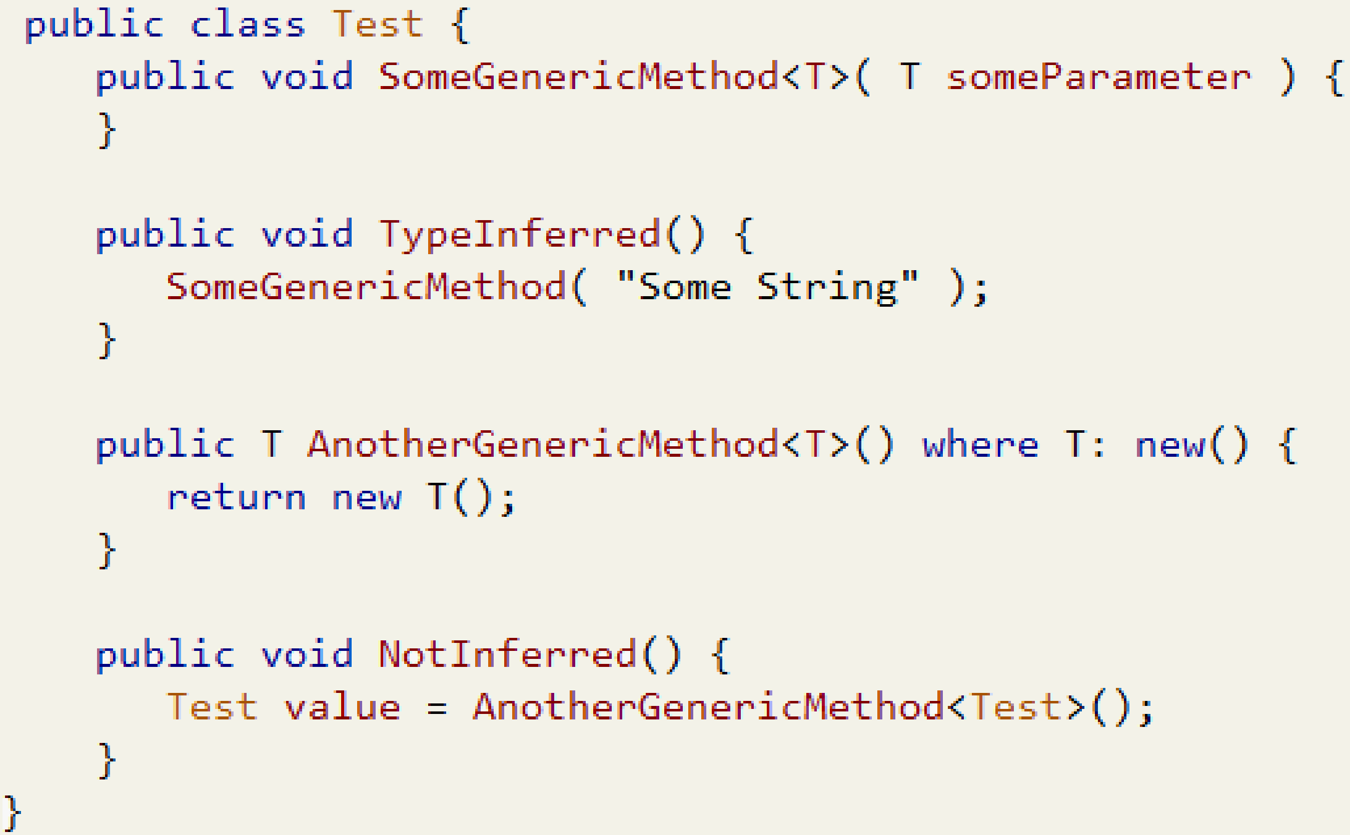
- Use == and != with caution
  - **Only does reference comparison**
- See UnconstrainedTests in sample code

IEnumerable Cast



```
ArrayList values = new ArrayList {"Apple", "Orange", "Kumquat"};

string result = values.Cast<string>().JoinWithPlus();
```

Type Inference



```
public class Test {
    public void SomeGenericMethod<T>( T someParameter ) {
    }

    public void TypeInferred() {
        SomeGenericMethod( "Some String" );
    }

    public T AnotherGenericMethod<T>() where T: new() {
        return new T();
    }

    public void NotInferred() {
        Test value = AnotherGenericMethod<Test>();
    }
}
```

Real World Examples

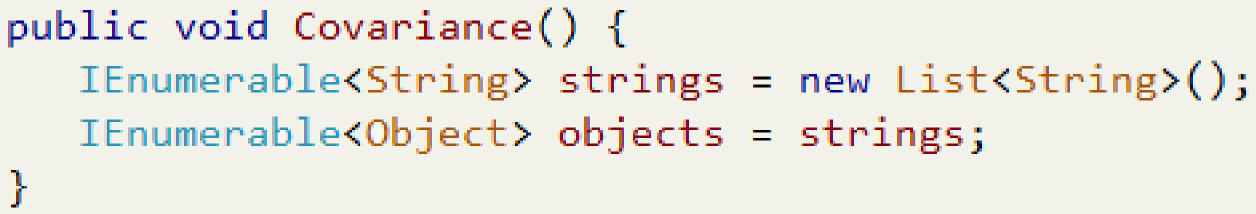
- Serialization
- Finder Tab
- Chart
- Filtering

Enum

- Can’t write a constraint to an Enum
- Constrain to struct, IConvertible
- Check for typeof(type).IsEnum
- Not compile time safe

Covariance/Contravariance

- Permits child to be used in place of parent class.
- Introduced with .NET 4



```
public void Covariance() {
    IEnumerable<String> strings = new List<String>();
    IEnumerable<Object> objects = strings;
}
```

Wirtley
Consulting LLC

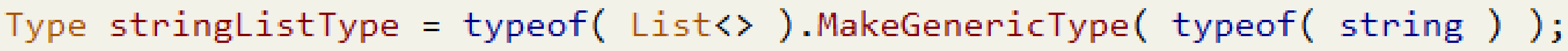
Terminology

- An *unbound* type has no type arguments specified
- A *constructed* type has at least one type argument specified
- A type parameter is an *open* type
- An *open constructed* type has at least one type argument which is an open type
- A *closed type* is any type which isn't open

http://bit.ly.com/GEN-Terms

Reflection

- Type.IsGenericType
- Interrogate
- Create



```
Type stringListType = typeof( List<> ).MakeGenericType( typeof( string ) );
```

Contact Me

GitHub: https://bit.ly/GenericsInNET

@JoeWirtley

http://WirtleyConsulting.com

Wirtley
Consulting LLC