Objects Comparer

# Introduction

Objects comparer is object-to-object comparer, which allows you recursively compare objects member by member and define comparison rules for certain properties, fields or types.

Objects comparer can be considered as ready to use framework or as an idea for similar solutions.

# Installation

Install-Package ObjectsComparer

# Basic Example

public class ClassA

{

public string StringProperty { get; set; }

public int IntProperty { get; set; }

# }

var a1 = new ClassA { StringProperty = "String", IntProperty = 1 };

var a2 = new ClassA { StringProperty = "String", IntProperty = 1 };

var comparer = new Comparer<ClassA>();

var isEqual = comparer.Compare(a1, a2);

Debug.WriteLine("a1 and a2 are " + (isEqual ? "equal" : "not equal"));

a1 and a2 are equal

var a1 = new ClassA { StringProperty = "String", IntProperty = 1 };

var a2 = new ClassA { StringProperty = "String", IntProperty = 2 };

var comparer = new Comparer<ClassA>();

IEnumerable<Difference> differenses;

var isEqual = comparer.Compare(a1, a2, out differenses);

var differensesList = differenses.ToList();

Debug.WriteLine("a1 and a2 are " + (isEqual ? "equal" : "not equal"));

if (!isEqual)

{

Debug.WriteLine("Differences:");

Debug.WriteLine(string.Join(Environment.NewLine, differensesList));

}

a1 and a2 are not equal

Differences:

Difference: MemberPath='IntProperty', Value1='1', Value2='2'

# Comparison Settings

RecursiveComparison. True by default.

EmptyAndNullEnumerablesEqual. False by default.

Comparison Settings allow to store custom values than can be used in custom comparers.

SetCustomSetting<T>(T value, string key = null)

GetCustomSetting<T>(string key = null)

# Overriding comparison rules

Comparer should be inherited from AbstractValueComparer<T> or implement IValueComparer<T>

public class MyComparer: AbstractValueComparer<string>

{

public override bool Compare(string obj1, string obj2, ComparisonSettings settings)

{

return obj1 == obj2; //Implement comparison logic here

}

# }

Type comparison rule override.

comparer.AddComparerOverride<string>(new MyComparer());

Field comparison rule override.

comparer.AddComparerOverride(() => new ClassA().StringProperty, new MyComparer());

# Factory

Factory should implement IComparersFactory or should be inherited from ComparersFactory.

public class MyComparersFactory: ComparersFactory

{

public override IComparer<T> GetObjectsComparer<T>(ComparisonSettings settings = null, IBaseComparer parentComparer = null)

{

if (typeof(T) == typeof(ClassA))

{

var comparer = new Comparer<ClassA>(settings, parentComparer, this);

comparer.AddComparerOverride<Guid>(new MyCustomGuidComparer());

return (IComparer<T>)comparer;

}

return base.GetObjectsComparer<T>(settings, parentComparer);

}

}

# Non-generic comparer

var comparer = new Comparer<ClassA>();

var isEqual = comparer.Compare(a1, a2);

This comparer creates generic implementation of comparer for each comparison.

# Predefined Value Comparers

There are some custom comparers that can be useful.

DoNotCompareValueComparer. Use to skip some fields/types. Has singleton implementation (DoNotCompareValueComparer.Instance).

DynamicValueComparer<T>. Receives comparison rule as function.

NulableStringsValueComparer. Null and empty strings are equal.

# Example 1: Expected Message

Challenge: Check if received message equal to the expected message.

public class Error

{

public int Id { get; set; }

public string Messgae { get; set; }

}

public class Message

{

public string Id { get; set; }

public DateTime DateCreated { get; set; }

public int MessageType { get; set; }

public int Status { get; set; }

public List<Error> Errors { get; set; }

public override string ToString()

{

return $"Id:{Id}, Date:{DateCreated}, Type:{MessageType}, Status:{Status}";

}

}

[TestFixture]

public class Example1Tests

{

private IComparer<Message> \_comparer;

[SetUp]

public void SetUp()

{

\_comparer = new Comparer<Message>(

new ComparisonSettings

{

//Null and empty error lists are equal

EmptyAndNullEnumerablesEqual = true

});

//Do not compare DateCreated

\_comparer.AddComparerOverride<DateTime>(DoNotCompareValueComparer.Instance);

//Do not compare Id

\_comparer.AddComparerOverride(() => new Message().Id, DoNotCompareValueComparer.Instance);

//Do not compare Message Text

\_comparer.AddComparerOverride(() => new Error().Messgae, DoNotCompareValueComparer.Instance);

}

[Test]

public void EqualMessagesWithoutErrorsTest()

{

var expectedMessage = new Message

{

MessageType = 1,

Status = 0,

};

var actualMessage = new Message

{

Id = "M12345",

DateCreated = DateTime.Now,

MessageType = 1,

Status = 0,

};

var isEqual = \_comparer.Compare(expectedMessage, actualMessage);

Assert.IsTrue(isEqual);

}

[Test]

public void EqualMessagesWithErrorsTest()

{

var expectedMessage = new Message

{

MessageType = 1,

Status = 1,

Errors = new List<Error>

{

new Error { Id = 2 },

new Error { Id = 7 }

}

};

var actualMessage = new Message

{

Id = "M12345",

DateCreated = DateTime.Now,

MessageType = 1,

Status = 1,

Errors = new List<Error>

{

new Error { Id = 2, Messgae = "Some error #2" },

new Error { Id = 7, Messgae = "Some error #7" },

}

};

var isEqual = \_comparer.Compare(expectedMessage, actualMessage);

Assert.IsTrue(isEqual);

}

}

# Example 2: Persons comparison

Challenge: Compare persons from different sources.

public class Person

{

public Guid PersonId { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public string MiddleName { get; set; }

public string PhoneNumber { get; set; }

public override string ToString()

{

return $"{FirstName} {MiddleName} {LastName} ({PhoneNumber})";

}

}

Phone number can have different formats. Let’s compare only digits.

public class PhoneNumberComparer: AbstractValueComparer<string>

{

public override bool Compare(string obj1, string obj2, ComparisonSettings settings)

{

return ExtractDigits(obj1) == ExtractDigits(obj2);

}

private string ExtractDigits(string str)

{

return string.Join(

string.Empty,

(str ?? string.Empty)

.ToCharArray()

.Where(char.IsDigit));

}

}

Factory allows not to configure comparer every time we need to create it.

public class MyComparersFactory: ComparersFactory

{

public override IComparer<T> GetObjectsComparer<T>(ComparisonSettings settings = null, IBaseComparer parentComparer = null)

{

if (typeof(T) == typeof(Person))

{

var comparer = new Comparer<Person>(settings, parentComparer, this);

//Do not compare PersonId

comparer.AddComparerOverride<Guid>(DoNotCompareValueComparer.Instance);

//Sometimes MiddleName can be skipped. Compare only if property has value.

comparer.AddComparerOverride(

() => new Person().MiddleName,

new DynamicValueComparer<string>(

(s1, s2, parentSettings) => string.IsNullOrWhiteSpace(s1) || string.IsNullOrWhiteSpace(s2) || s1 == s2,

s => s));

comparer.AddComparerOverride(

() => new Person().PhoneNumber,

new PhoneNumberComparer());

return (IComparer<T>)comparer;

}

return base.GetObjectsComparer<T>(settings, parentComparer);

}

}

[TestFixture]

public class Example2Tests

{

private MyComparersFactory \_factory;

private IComparer<Person> \_comparer;

[SetUp]

public void SetUp()

{

\_factory = new MyComparersFactory();

\_comparer = \_factory.GetObjectsComparer<Person>();

}

[Test]

public void EqualPersonsTest()

{

var person1 = new Person

{

PersonId = Guid.NewGuid(),

FirstName = "John",

LastName = "Doe",

MiddleName = "F",

PhoneNumber = "111-555-8888"

};

var person2 = new Person

{

PersonId = Guid.NewGuid(),

FirstName = "John",

LastName = "Doe",

PhoneNumber = "(111) 555 8888"

};

IEnumerable<Difference> differenses;

var isEqual = \_comparer.Compare(person1, person2, out differenses);

Assert.IsTrue(isEqual);

Debug.WriteLine($"Persons {person1} and {person2} are equal");

}

[Test]

public void DifferentPersonsTest()

{

var person1 = new Person

{

PersonId = Guid.NewGuid(),

FirstName = "Jack",

LastName = "Doe",

MiddleName = "F",

PhoneNumber = "111-555-8888"

};

var person2 = new Person

{

PersonId = Guid.NewGuid(),

FirstName = "John",

LastName = "Doe",

MiddleName = "L",

PhoneNumber = "222-555-9999"

};

IEnumerable<Difference> differenses;

var isEqual = \_comparer.Compare(person1, person2, out differenses);

var differensesList = differenses.ToList();

Assert.IsFalse(isEqual);

Assert.AreEqual(3, differensesList.Count);

Assert.IsTrue(differensesList.Any(d => d.MemberPath == "FirstName" && d.Value1 == "Jack" && d.Value2 == "John"));

Assert.IsTrue(differensesList.Any(d => d.MemberPath == "MiddleName" && d.Value1 == "F" && d.Value2 == "L"));

Assert.IsTrue(differensesList.Any(d => d.MemberPath == "PhoneNumber" && d.Value1 == "111-555-8888" && d.Value2 == "222-555-9999"));

Debug.WriteLine($"Persons {person1} and {person2}");

Debug.WriteLine("Differences:");

Debug.WriteLine(string.Join(Environment.NewLine, differensesList));

}

}

Persons John F Doe (111-555-8888) and John Doe ((111) 555 8888) are equal

Persons Jack F Doe (111-555-8888) and John L Doe (222-555-9999)

Differences:

Difference: MemberPath='FirstName', Value1='Jack', Value2='John'.

Difference: MemberPath='MiddleName', Value1='F', Value2='L'.

Difference: MemberPath='PhoneNumber', Value1='111-555-8888', Value2='222-555-9999'.