

همه چیز در خصوص مقایسه و برابری در سی شارپ

مهدی کیانی

مقدمه

یکی از مفاهیم مهم در زبان های برنامه نویسی مفهوم برابری اشیا می باشد. به عنوان مثال در شبه کد زیر

```
DataType variable1 = value1;  
DataType variable2 = value2;
```

چه زمانی متغیر های variable1 و variable2 با هم برابرند؟ آیا این موضوع به مقادیر آن ها یعنی value1 و value2 بستگی دارد؟ آیا به نوع داده آن ها یعنی DataType1 و DataType2 و یا دیگر شرایط؟

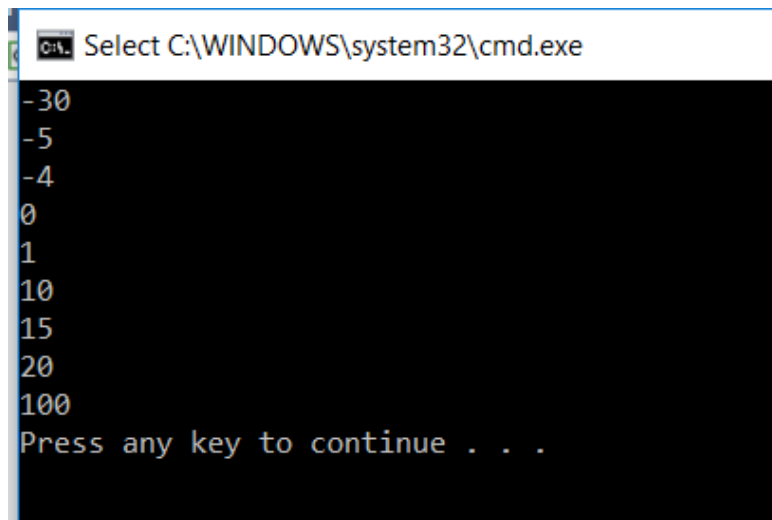
مشخصا نمی توان پاسخ واحدی به این سوالات برای همه زبان های برنامه نویسی ارائه داد. چراکه این موضوع به شرایط مختلفی از جمله انواع داده ها و نحوه ذخیره سازی آن ها در حافظه، روش های مقایسه و ... بستگی دارد. در این مقاله مشخصا به پاسخ این سوالات در زبان سی شارپ و پیاده سازی های ژنریک پرداخته خواهد شد.

اینترفیس IComparable

دستورات زیر را در نظر بگیرید:

```
int[] data = {1,10,-5,20,-30,0,-4,100,15 };  
  
Array.Sort(data);  
  
foreach (var item in data)  
{  
    Console.WriteLine(item);  
}
```

خروجی این برنامه به صورت زیر است:



```
Select C:\WINDOWS\system32\cmd.exe
-30
-5
-4
0
1
10
15
20
100
Press any key to continue . . .
```

حال یک کلاس به نام Person و به صورت زیر تعریف کنید:

```
public class Person
{
    public int Age { get; set; }
}
```

سپس دستورات زیر را نوشته و اجرا کنید:

```
Person[] persons =
{
    new Person { Age =20 },
    new Person { Age =30 },
    new Person { Age =10 },
    new Person { Age =28 },
    new Person { Age =12 },
    new Person { Age =32 }
};
Array.Sort(persons);
foreach (var person in persons)
{
    Console.WriteLine(person);
}
```

در دستورات فوق یک آرایه از نوع Person تعریف شده و سعی شده است تا مانند روش قبل از متد Sort کلاس Array استفاده گردد.

اگر دستورات فوق را اجرا کنید با خطای زیر روبرو خواهید شد :

Exception Unhandled



System.InvalidOperationException: 'Failed to compare two elements in the array.'

همانطور که در تصویر مشخص است، متد `Array` نتوانسته است که عناصر درون آرایه را با یکدیگر مقایسه کند و به همین دلیل نمی داند که چگونه و بر چه اساسی باید عملیات مرتب سازی را انجام دهد چراکه مقایسه داده ها پایه و اساس مرتب سازی آن ها است.

اگر به شماتیک متد `Sort` (با امضایی که در دستورات فوق به کار رفته است) در کلاس `Array` مراجعه کنیم، توضیح زیر را در خصوص عملکرد این متد مشاهده خواهیم کرد :

Sorts the elements in an entire `System.Array` using the `System.IComparable<T>` generic interface implementation of each element of the `System.Array`.

یکی از روش های که متد `Sort` می تواند بر اساس آن داده های یک آرایه را مرتب سازی کند، استفاده از اینترفیس `System.IComparable` (نوع ژنریک آن در توضیح فوق) می باشد.

تعریف این اینترفیس به صورت زیر است :

```
public interface IComparable
{
    int CompareTo (object obj);
}
```

ونوع ژنریک آن :

```
public interface IComparable<in T>
{
    int CompareTo (T other);
}
```

یکی از روش هایی که متد Sort از کلاس Array برای مرتب سازی آرایه ها استفاده می کند استفاده از اینترفیس فوق می باشد. در واقع متد Sort براساس فراخوانی این متد که برای نوع داده درون آرایه پیاده سازی شده است و خروجی حاصل از فراخوانی آن می توان ترتیب عناصر را مشخص کند. بنابر این نوع داده درون آرایه می بایستی اینترفیس مذکور را پیاده سازی کرده باشد تا بتوان آرایه ای از آن را توسط متد Sort از کلاس Array مرتب کرد.

اینترفیس فوق برای برخی از انواع داده به صورت پیش فرض پیاده سازی شده است. نوع داده int یکی از این انواع است. اگر به شماتیک این نوع داده نگاه کنیم، بخشی از آن به صورت زیر خواهد بود :

```
public struct Int32: IComparable<Int32>
```

بنابر این متد Sort نحوه مرتب سازی آرایه ای از int ها را می داند. چراکه اینترفیس IComparable برای آن پیاده سازی شده است.

اما در خصوص کلاس Person مسئله چگونه است؟ همانطور که در دستورات این کلاس قبل تر مشاهده کردید، اینترفیس مذکور پیاده سازی نشده است و به همین جهت متد Sort نحوه مقایسه دو نمونه از کلاس Person را نمی داند و بنابر این عملیات مرتب سازی با خطا روبرو شده است.

دستورات کلاس Person را به صورت زیر تغییر دهید :

```
public class Person : IComparable<Person>
{
    public int Age { get; set; }

    public int CompareTo(Person other)
    {
        return Age.CompareTo(other.Age);
    }
}
```

حال دستورات زیر را مجددا اجرا کنید :

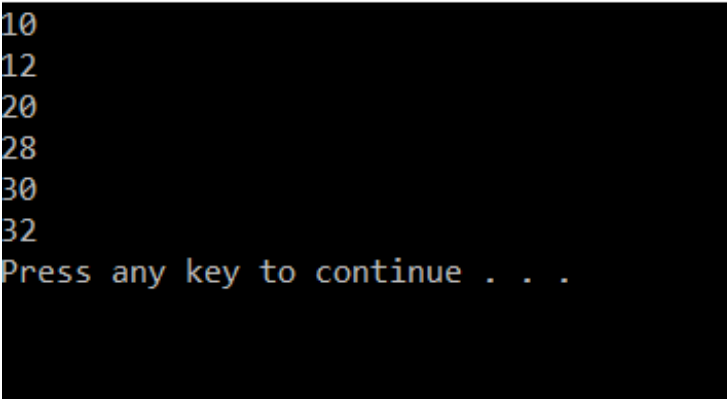
```
Person[] persons =
{
    new Person { Age =20 },
    new Person { Age =30 },
    new Person { Age =10 },
    new Person { Age =28 },
}
```

```

    new Person { Age = 12 },
    new Person { Age = 32 }
};
Array.Sort(persons);
foreach (var person in persons)
{
    Console.WriteLine(person);
}

```

خروجی دستورات فوق شبیه به زیر خواهد بود :



```

C:\WINDOWS\system32\cmd.exe
10
12
20
28
30
32
Press any key to continue . . .

```

همانطور که مشاهده می کنید، در این حالت برنامه بدون خطا اجرا شده و داده های درون آرایه مرتب می شوند.

اینترفیس IComparer

یکی دیگر از سربرگزاری های متد Sort از کلاس Array به صورت زیر می باشد:

```

public static void Sort<T>(T[] array, IComparer<T> comparer);

```

در این سربرگزاری در پارامتر دوم از اینترفیس IComparer به صورت ژنریک استفاده شده است. در واقع در پارامتر دوم این متد ی بایستی نمونه ای از کلاسی که اینترفیس IComparer را پیاده سازی کرده است ارسال کنیم.

اما اینترفیس IComparer چیست؟

اگر به تعریف این اینترفیس نگاهی بیاندازیم، خواهیم داشت :

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

اینترفیس IComparer دارای یک متد به نام Compare با دو آرگومان ورودی و نوع خروجی int می باشد. نوع آرگومان ورودی در زمان پیاده سازی اینترفیس مشخص می شود. مقدار خروجی این متد یک عدد صحیح علامت دارد است. مقدار کمتر از 0 به معنی کوچکتر بودن X از Y، مقدار 0 به معنی برابری X و Y و مقدار بزرگتر از 0 به معنی بزرگتر بودن X از Y است.

متد Sort می تواند با فراخوانی این متد عملیات مرتب سازی را انجام دهد.

کلاس Person را به صورت زیر تغییر دهید :

```
public class Person : IComparer<Person>
{
    public int Age { get; set; }

    public int Compare(Person x, Person y)
    {
        return x.Age.CompareTo(y.Age);
    }
}
```

همانطور که مشاهده می کنید اینترفیس IComparer با نوع داده Person برای کلاس Person پیاده سازی شده است. متد Compare، خواص Age دو نمونه از کلاس Person را با یکدیگر مقایسه می کند.

حال دستورات زیر را اجرا کنید :

```
Person[] persons =
{
    new Person { Age = 20 },
    new Person { Age = 30 },
    new Person { Age = 10 },
    new Person { Age = 28 },
    new Person { Age = 12 },
    new Person { Age = 32 }
};
Array.Sort(persons, new Person());
foreach (var person in persons)
{
    Console.WriteLine(person.Age);
}
```

به دستور هایلایت شده توجه کنید. پارامتر دوم متد Sort یک نمونه از کلاس Person می باشد. چرا که پارامتر دوم Sort باید نمونه ای از کلاسی باشد که اینترفیس IComparer را (برای نوع داده های آرایه) پیاده سازی کرده است و چون این عمل را در دستورات قبل برای کلاس Person انجام دادیم، لذا نمونه ای از کلاس Person برای عملیات مقایسه دو نمونه از کلاس Person استفاده شده است.

بهبود استفاده از اینترفیس IComparer

به جهت عدم وابستگی کد ها به یکدیگر و قابلیت توسعه پذیری بهتر، مرسوم نیست که از روش فوق (پیاده سازی اینترفیس IComparer در خود کلاس مورد مقایسه) استفاده گردد و بهتر است برای پیاده سازی این اینترفیس از کلاس دیگری استفاده شود. این کار هم باعث می شود تا وابستگی بین کد ها از بین رفته و هم بتوان کنترل بیشتری بر روی روش های مقایسه دو نمونه از یک نوع داده اعمال کرد.

توجه داشته باشید که الزاما قرار نیست ما بر روی کلاس Person عملیات مقایسه برای مرتب سازی داشته باشیم لذا گره زدن دستورات مقایسه با خود کلاس روش مناسبی نیست. در ادامه روش بهبود این مسئله را با هم خواهیم دید.

ابتدا کلاس Person را به حالت قبل خود بازگردانید:

```
public class Person
{
    public int Age { get; set; }
}
```

یک کلاس به نام PersonComparer به صورت زیر تعریف کنید :

```
public class PersonComparer: IComparer<Person>
{
    public int Compare (Person x, Person y)
    {
        return x.Age.CompareTo(y.Age);
    }
}
```

حال دستورات متد Sort را به صورت زیر تغییر دهید :

```
...
Array. Sort(people, new PersonComparer());
...
```

توجه داشته باشید که در این حالت، عملیات مقایسه و مرتب سازی مستقل از کلاس Person بوده که موجب کاهش وابستگی کد ها می شود.

توسعه پذیری

یکی از مزایای جدا سازی تعریف یک نوع و تعریف عملیات مقایسه آن، توسعه پذیری عملیات مقایسه می باشد. اجازه دهید مطلب را با یک مثال مشاهده کنیم:

ابتدا کلاس Person را بسط داده و به صورت زیر تغییر دهید :

```
public class Person
{
    public int Age { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }

    public override string ToString()
    {
        return $"{FirstName},{LastName},{Age}";
    }
}
```

حال یک نوع داده شمارشی به نام PersonCompareMode به صورت زیر تعریف کنید :

```
public enum PersonCompareMode
{
    ByAge,
    ByFirstName,
    ByLastName
}
```

سپس کلاس PersonComparer به صورت زیر تغییر دهید:

```
public class PersonComparer : IComparer<Person>
{
    private readonly PersonCompareMode compareMode;

    public PersonComparer(PersonCompareMode compareMode)
    {
        this.compareMode = compareMode;
    }
    public int Compare(Person x, Person y)
    {
        switch (compareMode)
        {
            case PersonCompareMode.ByAge:
                return x.Age.CompareTo(y.Age);
            case PersonCompareMode.ByFirstName:
                return x.FirstName.CompareTo(y.FirstName);
            case PersonCompareMode.ByLastName:
```



```

        return x.LastName.CompareTo(y.LastName);
    default:
        throw new InvalidOperationException("Invalid compare mode!");
    }
}
}

```

در تغییرات فوق یک نوع داده شماری به کانستراکتور کلاس `PersonComparer` ارسال شده است و سپس بر اساس مقدار این فیلد در متد `Compare` عملیات مقایسه انجام شده است.

حال دستورات تعریف آرایه را به صورت زیر تغییر دهید :

```

Person[] persons =
{
    new Person { Age =20,FirstName ="Mehdi",LastName="Kiani" },
    new Person { Age =30 ,FirstName ="Ali",LastName="Hasani"},
    new Person { Age =10 ,FirstName ="Bagher",LastName="Fatemi"},
    new Person { Age =28 ,FirstName ="Sara",LastName="Amiri"},
    new Person { Age =12 ,FirstName ="Yasin",LastName="Mohamadi"},
    new Person { Age =32 ,FirstName ="Zahra",LastName="Naseri"}
};

```

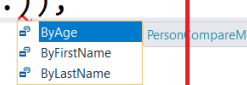
شما می توانید داده های خود را به جای داده های فوق قرار دهید.

حال موقع فراخوانی متد `Sort` و استفاده از کلاس `PersonComparer` می توانید نوع مقایسه و درواقع نوع مرتب سازی را تعیین کنید :

```

Array.Sort(persons,new PersonComparer(PersonCompareMode.));

```



همانطور که مشاهده می کنید می توانید بر حسب `Age` ، `FirstName` و یا `LastName` اقدام به مرتب سازی نمائید.

مثال 1) مرتب سازی بر اساس سن)

```

Array.Sort(persons,new PersonComparer(PersonCompareMode.ByAge));
foreach (var person in persons)
{
    Console.WriteLine(person);
}

```

```
}
```

خروجی مثال 1:

```
C:\WINDOWS\system32\cmd.exe
Bagher,Fatemi,10
Yasin,Mohamadi,12
Mehdi,Kiani,20
Sara,Amiri,28
Ali,Hasani,30
Zahra,Naseri,32
Press any key to continue . . .
```

مثال 2: (مرتب سازی بر اساس نام)

```
Array.Sort(persons,new PersonComparer(PersonCompareMode.ByFirstName));
foreach (var person in persons)
{
    Console.WriteLine(person);
}
```

خروجی مثال 2:

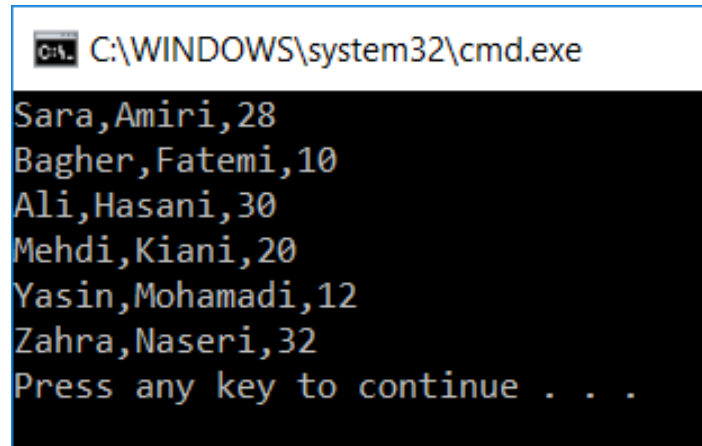
```
C:\WINDOWS\system32\cmd.exe
Ali,Hasani,30
Bagher,Fatemi,10
Mehdi,Kiani,20
Sara,Amiri,28
Yasin,Mohamadi,12
Zahra,Naseri,32
Press any key to continue . . .
```

مثال 3: (مرتب سازی بر اساس نام خانوادگی)

```
Array.Sort(persons,new PersonComparer(PersonCompareMode.ByLastName));
foreach (var person in persons)
```

```
{  
    Console.WriteLine(person);  
}
```

خروجی مثال 3:



```
C:\WINDOWS\system32\cmd.exe  
Sara,Amiri,28  
Bagher,Fatemi,10  
Ali,Hasani,30  
Mehdi,Kiani,20  
Yasin,Mohamadi,12  
Zahra,Naseri,32  
Press any key to continue . . .
```

مقایسه با استفاده از دلیگیت Comparison

یکی دیگر از ابزار های موجود برای مقایسه دو نمونه از یک نوع داده در سی شارپ استفاده از دلیگیت Comparison می باشد. تعریف این دلیگیت به صورت زیر است :

```
public delegate int Comparison<in T>(T x, T y);
```

این دلیگیت می تواند به هر متدی با امضای مشابه به امضای دلیگیت اشاره و آن را فراخوانی کند.

یکی دیگر از سربارگزاری های متد Sort استفاده از این دلیگیت برای عملیات مقایسه است :

```
public static void Sort<T>(T[] array, Comparison<T> comparison);
```

همانطور که مشاهده می کنید آرگومان دوم از جنس دلیگیت مذکور است.

به عنوان مثال متد زیر را در نظر بگیرید :

```
private static int ComparePerson(Person x, Person y)  
{  
    return x.Age.CompareTo(y.Age);  
}
```

کاربرد متد فوق شبیه به پیاده سازی های قبلی است که تاکنون مشاهده کردید. استفاده از کلمه static ضرورتی نداشته و در اینجا با فرض اینکه متد مذکور در یک متد static فراخوانی می شود (مثلا متد Main کلاس Program پروژه های کنسولی) از این کلمه در تعریف متد استفاده شده است.

حال می توانید با استفاده از متد Sort و نیز استفاده از متد فوق به عنوان مقایسه کننده استفاده کنید و آرایه ای از Person ها را مرتب سازی کنید. برای این منظور کافیست که متد Sort را به صورت زیر فراخوانی کنید:

```
Array.Sort(persons, ComparePerson);
```

آرگومان دوم نام متدی است که عملیات مقایسه را انجام می دهد (در اینجا مرتب سازی بر اساس Age انجام خواهد شد)

شما می توانید برای حالت های مختلف متد های مختلف بنویسید.

استفاده از این روش در زمانی که بخواهید عملیات مقایسه را شخصی سازی کنید و به و صورت موردی از آن استفاده کنید می تواند مفید باشد.

همچنین می توانید به جای تعریف متد با نام (مانند ComparePerson در دستورات فوق) از متد های بی نام استفاده کنید.

به عنوان مثال می توانید متد Sort را بدون نیاز به متد ComparePerson و به صورت زیر استفاده نمائید :

```
Array.Sort(persons, delegate(Person x, Person y) { return x.Age.CompareTo(y.Age); });
```

یا از عبارات لامبدا برای این منظور استفاده کنید :

```
Array.Sort(persons, (x, y) => x.Age.CompareTo(y.Age));
```

توجه : استفاده از کلاس Array و متد Sort برای بیان مفاهیم گفته شده در فوق، بدین معنی نیست که اینترفیس های Comparable و Comparer و دلیگیت Comparison صرفا باید در متد Sort کلاس Array استفاده شوند. شما می توانید از این مطالب در هر نوع پیاده سازی مد نظر خود بهره برداری کنید. در اینجا صرفا به جهت سادگی و اینکه متد Sort کلاس Array از همه موارد گفته شده پشتیبانی می کند، از این کلاس استفاده شده است.

کلاس Object

در سی شارپ تمامی انواع یک Object بوده به صورت مستقیم یا غیر مستقیم از کلاس Object مشتق می شوند حتی انواع مقداری.

کلاس Object دارای 7 متد به صورت زیر می باشد :

```
public static bool Equals(Object objA, Object objB);  
public static bool ReferenceEquals(Object objA, Object objB);  
public virtual bool Equals(Object obj);  
public virtual int GetHashCode();  
public Type GetType();  
public virtual string ToString();  
protected Object MemberwiseClone();
```

متد MemberwiseClone

برای پیاده سازی یک کپی کم سطح (Shallow Copy) از شی به کار می رود. که بحث بر روی آن خارج از موضوع این مقاله است.

متد ToString

این متد نمایش رشته ای پیش فرض شی مورد نظر را پیاده سازی می کند. این متد قابل دوباره نویسی بوده که یک نمونه آن را در کلاس Person و در دستورات قبل مشاهده کردید.

متد GetType

این متد یک نمونه از کلاس Type که بیان گر اطلاعات مربوط به شی است (نوع داده شی، نام آن، فضای نام و ...) را بر می گرداند. از این متد در مباحث reflection مورد استفاده قرار می گیرد که بحث بر روی آن خارج از موضوع این مقاله است. فقط در اینجا به جهت نمایش این موضوع که هر نوعی در واقع یک Object است به مثال می آورم.

دستورات زیر را اجرا کنید :

```
Person p = new Person();  
Type pt = p.GetType();  
Console.WriteLine("p(Person) is object ? {0}", p is object);  
Console.WriteLine("p(Person) base type : {0} ", pt.BaseType);  
Console.WriteLine("p(Person) type name : {0} ", pt.Name);
```

```

Console.WriteLine("=====");
int a = 10;
Type at = a.GetType();

Console.WriteLine("a(int) is object ? {0}", a is object);
Console.WriteLine("a(int) base type : {0} ", at.BaseType);
Console.WriteLine("a(int) base type of base type : {0} ", at.BaseType.BaseType);
Console.WriteLine("a(int) type name : {0} ", at.Name);

```

خروجی دستورات فوق در شل زیر نشان داده شده است:

```

p(Person) is object ? True
p(Person) base type : System.Object
p(Person) type name : Person
=====
a(int) is object ? True
a(int) base type : System.ValueType
a(int) base type of base type : System.Object
a(int) type name : Int32

```

همانطور که در تصویر مشخص است، نوع داده Person و int مورد بررسی قرار گرفته اند. نوع Person یک کلاس و از نوع ارجاعی و نوع int یک نوع داده اولیه مقداری است. هر دو نوع در نهایت یک Object می باشند. توجه کنید که نوع داده int با یک واسطه به نوع داده Object می رسد.

متد های Equals در کلاس Object

در کلاس Object سه متد به منظور بررسی برابری شی تعریف شده است که بررسی آن ها بخشی از مطالب مربوط به این مقاله می باشد.

متد ReferenceEquals

این متد که دو آرگومان ورودی از نوع Object دریافت می کند بررسی می کند که آیا نمونه های هر شی یکسان هستند یا خیر. چنانچه نمونه های هر دو شی یکسان باشند و یا هر دو آرگومان ورودی null باشند، این متد مقدار true و در غیر اینصورت مقدار false بر می گرداند. توجه داشته باشید که این متد به صورت static تعریف شده است.

به دستورات زیر دقت کنید :

```
Person p1 = null;
Person p2 = null;
Person p3 = new Person();
Person p4 = new Person();
Console.WriteLine("p1 equals p2 ? {0}", object.ReferenceEquals(p1, p2));
Console.WriteLine("p1 equals p3 ? {0}", object.ReferenceEquals(p1, p3));
Console.WriteLine("p3 equals p4 ? {0}", object.ReferenceEquals(p3, p4));
p4 = p3;
Console.WriteLine("p3 equals p4 ? {0}", object.ReferenceEquals(p3, p4));
```

خروجی دستورات فوق به صورت زیر می باشد:

```
p1 equals p2 ? True
p1 equals p3 ? False
p3 equals p4 ? False
p3 equals p4 ? True
```

متغیر p1 و p2 هر دو به صورت null تعریف شده اند. به همین دلیل خط اول خروجی نتیجه true را نشان می دهد.

متغیر p3 دارای یک نمونه از کلاس Person می باشد. بنابر این خط دوم در خروجی که برابری p1 و p3 را بررسی می کند نتیجه false بر می گرداند.

متغیر p4 نیز همانند متغیر p3 با یک نمونه از کلاس Person مقدار دهی شده است. واضح است که نمونه های ایجاد شده برای p3 و p4 متفاوت هستند لذا خط سوم خروجی مقدار false را نشان می دهد.

در دستور هایلایت شده، متغیر p4 برابر با متغیر p3 قرار داده شده است، بنابر این در این حالت ارجاع های p3 و p4 به یک شی بوده و خط چهارم خروجی مقدار true را بر می گرداند.

خروجی متد مذکور ممکن است برای دو حالت زیر گیج کننده باشد.

✓ مقایسه دو نوع داده مقداری

✓ مقایسه دو رشته

حالت اول : دستورات زیر را در نظر بگیرید :

```
int a = 10;
Console.WriteLine("a is equal to a? {0}", object.ReferenceEquals(a, a));
```

دستور فوق توسط متد `ReferenceEquals` متغیر `a` را با خودش مقایسه می کند. اگر دستور فوق را اجرا کنید، بر خلاف تصور خروجی `false` خواهد بود. این اتفاق در مورد انواع مقداری و به دلیل عملیات `Boxing` در هنگام ارسال متغیر `a` به متد `ReferenceEquals` رخ می دهد.

حالت دوم : دستورات زیر را در نظر بگیرید :

```
string s1 = "Mehdi";
string s2 = "Mehdi";
Console.WriteLine("s1 is equal to s2? {0}", object.ReferenceEquals(s1,s2));
```

همانطور که مشاهده می کنید متغیرهای رشته ای `s1` و `s2` به صورت مجزا از یکدیگر تعریف شده اند، اما بر خلاف تصور خروجی دستورات فوق `true` خواهد بود. دلیل این موضوع این است که CLR به صورت خودکار جدولی را تحت عنوان استخر داخلی نگهداری می کند. وقتی رشته ثابتی در برنامه تعریف می شود، درون این استخر قرار می گیرد که اصطلاحاً به آن `Intern` شدن می گویند. در دستورات فوق رشته `Mehdi` به استخر رشته ها `Intern` شده است. چنانچه متغیر رشته ای با یک مقدار تعریف شود، ابتدا در مورد داده مورد نظر جستجو می شود و اگر داده (رشته) مورد نر در استخر وجود داشته باشد ارجاع یکسانی به همان رشته برگشت داده می شود. در واقع اگر شما یک ثابت رشته ای را به چندین متغیر نسبت دهید، تمامی متغیرها به همان ثابت رشته ای اشاره کرده و از ایجاد ارجاع ها با مقادیر مختلف جلوگیری خواهد شد. بنابر این `s1` و `s2` هر دو به یک محل اشاره خواهند کرد و به همین جهت خروجی دستورات فوق `true` می باشد.

حال دستورات زیر را در ادامه دستورات فوق در نظر بگیرید :

```
string s3 = "Mehdi";
Console.WriteLine("s1 is equal to s3? {0}", object.ReferenceEquals(s1, s3));
Console.WriteLine("s2 is equal to s3? {0}", object.ReferenceEquals(s2,s3));
```

خروجی دو دستور فوق نیز `true` خواهد بود. چرا که `s3` نیز به رشته `Mehdi` که در استخر رشته ها `Intern` شده است اشاره می کند. کلاس `String` دارای متدی به نام `IsInterned` دارد که ارجاعی را به یک ثابت رشته ای (چنانچه در استخر وجود داشته باشد) بر می گرداند و در غیر این صورت مقدار `null` بر می گرداند.

دستورات زیر را در ادامه دستورات فوق در نظر بگیرید:

```
string s4 = "kiani";
s1 += s4;
s2 += s4;
Console.WriteLine("s1 is equal to s2? {0}", object.ReferenceEquals(s1, s2));
Console.WriteLine("s1 is interned? {0}", string.IsInterned(s1) != null);
```



```
Console.WriteLine("s1 is interned? {0}", string.IsInterned(s2) != null);
```

با وجود اینکه متغیرهای s1 و s2 هر دو دارای مقدار یکسانی می باشند، بر خلاف تصور نتیجه متد ReferenceEquals بر روی این دو متغیر مقدار false را بر می گرداند. علت این موضوع این است که مقادیر این متغیرها در استخر داخلی CLR وجود نداشته و بنابر این ارجاع های آن ها یکسان نمی باشد. این موضوع در خروجی دو دستور آخر قابل مشاهده است. متد IsInterned برای s1 و s2 مقدار null بر می گرداند.

متد Equals(Object objA, Object objB)

یکی دیگر از متدهای استاتیک کلاس Object متد Equals با دو پارامتر ورودی است. این متد یکسان بودن دو شی را با توجه به شرایط زیر بررسی کرده و نتیجه true یا false بر می گرداند.

✓ اگر هر دو پارامتر ارسال شده به این متد دارای ارجاع یکسان باشند، در این صورت نتیجه بررسی true خواهد بود. این مورد شبیه به استفاده از متد ReferenceEquals خواهد بود.

✓ اگر دو پارامتر ارسال شده به متد null باشند، نتیجه بررسی true خواهد بود.

✓ اگر یکی از پارامترهای ارسال شده به متد null باشد، نتیجه بررسی false خواهد بود.

✓ اگر هیچ یک از شرایط فوق برقرار نباشد، به این معنی که هیچ کدام از پارامترها null نبوده و هیچ کدام

نیز دارای ارجاع یکسان نباشند، آنگاه نتیجه این متد وابسته به فراخوانی متد دیگری به نام Equals می

باشد. متد مذکور بر خلاف دو متد قبلی تک پارامتری بوده و به صورت استاتیک نیز تعریف نشده است.

در این حالت نتیجه دستور objA.Equals(objB) به عنوان خروجی دستور

object.Equals(objA,objB) برگشت داده خواهد شد.

دستورات زیر را در نظر بگیرید:

```
int a = 10;
Console.WriteLine("Object.Equals(a,a) ? {0}", object.Equals(a, a));
Console.WriteLine("Object.ReferenceEquals(a,a) ? {0}", object.ReferenceEquals(a, a));
```

خروجی دستورات فوق به صورت زیر خواهد بود:

```
Object.Equals(a,a) ? True
Object.ReferenceEquals(a,a) ? False
a.Equals(a) ? True
```

همانطور که مشاهده می کنید، خروجی دستور اول true و خروجی دستور دوم false می باشد. علت خروجی دستور دوم را قبلاً بررسی کردیم. اما در خصوص دستور اول :

برای این دستور حالت چهارم از شرایط چهارگانه ای که در ابتدای توضیحات این متد گفته شد، برقرار است. یعنی هیچ یک از پارامترهای ارسال شده به متد null نبوده و علاوه بر این ارجاع های آن ها نیز یکسان نمی باشد. بنابر این خروجی دستور object.Equals(a,a) وابسته به خروجی دستور a.Equals(a) خواهد بود(این متد در بخش بعدی بررسی قرار می گیرد) . همانطور که مشاهده می کنید خروجی دستور a.Equals(a) برابر true بوده و در نتیجه خروجی دستور object.Equals(a,a) نیز برابر با true خواهد بود.

حال دستورات زیر را در نظر بگیرید:

```
Person p1 = new Person { Age = 10, FirstName = "Mehdi", LastName = "Kiani" };
Person p2 = new Person { Age = 10, FirstName = "Mehdi", LastName = "Kiani" };
Console.WriteLine("Object.Equals(p1,p2) ? {0}", object.Equals(p1, p2));
Console.WriteLine("Object.ReferenceEquals(p1,p2) ? {0}", object.ReferenceEquals(p1, p2));
Console.WriteLine("p1.Equals(p2) ? {0}", p1.Equals(p2));
```

خروجی دستورات فوق به صورت زیر می باشد.

```
Object.Equals(p1,p2) ? False
Object.ReferenceEquals(p1,p2) ? False
p1.Equals(p2) ? False
```

قبل از رفتن به بخش بعدی سعی کنید همانند مورد قبلی، خروجی های فوق را تفسیر کنید.

متد (obj) Equals(Object)

یکی دیگر از متد های کلاس Object متد Equals با یک ورودی می باشد. این متد بر خلاف متد قبلی به صورت استاتیک تعریف نشده است. اگر به حالت چهارم شروط چهارگانه ای که در متد قبلی بیان شد مجددا نگاه کنید، در صورتی که پارامتر های objA و objB دارای ارجاع های یکسان نبوده هیچ یک از آنها null نیز نباشند، آنگاه نتیجه دستور Object.Equals(objA,objB) وابسته به دستور objA.Equals(objB) خواهد بود. متد Equals در این حالت به صورت virtual تعریف شده است. پس بنابر این می توان آن را برای انواع مختلف دوباره نویسی کرد. عملی که برای انواع از پیش تعریف شده مانند int انجام شده است.

اما سوال این است که چه نیازه به دوباره نویسی این متد در انواع دیگر است؟ دستورات زیر را در نظر بگیرید :

```
Person p1 = new Person { Age = 10, FirstName = "Mehdi", LastName = "Kiani" };
Person p2 = new Person { Age = 10, FirstName = "Ali", LastName = "Hasani" };
Console.WriteLine("Object.Equals(p1,p2) ? {0}", object.Equals(p1, p2));
Console.WriteLine("Object.ReferenceEquals(p1,p2) ? {0}", object.ReferenceEquals(p1, p2));
Console.WriteLine("p1.Equals(p2) ? {0}", p1.Equals(p2));
```

خروجی دستورات فوق false می باشد(چرا؟)

```
Object.Equals(p1,p2) ? False
Object.ReferenceEquals(p1,p2) ? False
p1.Equals(p2) ? False
```

حال فرض کنید بخواهیم در برنامه شرایطی را برقرار کنیم که دو نمونه از کلاس Person براساس مقدار خاصیت Age بررسی شوند و چنانچه دو نمونه از کلاس Person دارای مقدار Age یکسان باشند، به عنوان دو نمونه یکسان در نظر گرفته شوند. در این حالت دوباره نویسی متد مذکور بسیار کارا می باشد.

دستورات کلاس Person را به صورت زیر تغییر دهید :

```
public class Person
{
    public int Age { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }

    public override bool Equals(object obj)
    {
        if (obj is null)
            return false;
        Person other = obj as Person;
```

```

        if (other == null)
            return false;
        return this.Age == other.Age;
    }

    public override string ToString()
    {
        return $"{FirstName},{LastName},{Age}";
    }
}

```

در دستورات فوق متد Equals دوباره نویسی شده و تنها در صورتی که پارامتر ورودی از نوع Person بوده و null نیز نباشد و مقدار Age آن با Age فعلی یکسان باشد مقدار true و در غیر این صورت مقدار false برگشت داده می شود.

حال دستورات زیر را اجرا کنید :

```

Person p1 = new Person { Age = 10, FirstName = "Mehdi", LastName = "Kiani" };
Person p2 = new Person { Age = 10, FirstName = "Ali", LastName = "Hasani" };
onsole.WriteLine("Object.Equals(p1,p2) ? {0}", object.Equals(p1, p2));
Console.WriteLine("Object.ReferenceEquals(p1,p2) ? {0}", object.ReferenceEquals(p1, p2));
Console.WriteLine("p1.Equals(p2) ? {0}", p1.Equals(p2));

```

خروجی دستورات فوق در شکیب زیر نشان داده شده است:

```

Object.Equals(p1,p2) ? True
Object.ReferenceEquals(p1,p2) ? False
p1.Equals(p2) ? True

```

اگر این خروجی را با خروجی قبلی مقایسه کنید، متوجه می شوید که خروجی خطوط اول و سوم با هم متفاوت است.

در حالت دوم، نتیجه اجرای دستور Object.Equals(p1,p2) به صورت زیر بررسی می شود :

✓ هیچ یک از دو پارامتر دارای مقدار null نمی باشند.

✓ دو پارامتر دارای ارجاع برابر نیستند (خروجی خط دوم)

بنابر شرایط فوق، حالت چهارم از شروط چهارگانه برقرار بود و لذا نتیجه خروجی دستور اول وابسته به نتیجه خروجی دستور `p1.Equals(p2)` می باشد. از آنجا که عملکرد این متد در کلاس `Person` دوباره نویسی شده است (که توضیح آن قبلاً داده شد) و با توجه به مقادیر یکسان خاصیت `Age` در هر دو نمونه `p1` و `p2` خروجی این دستور `true` بوده و بنابر این خروجی خط اول نیز `true` خواهد بود.

اینکه در چه شرایطی باید متد مذکور را در نوع داده ای خود دوباره نویسی کنیم خود مبحثی مفصل بوده که از حوصله این مقاله خارج است.

متد GetHashCode

آخرین متدی که از کلاس `Object` و همچنین به عنوان آخرین مطلب این مقاله مورد بررسی قرار می گیرد متد `GetHashCode` می باشد. این متد در هنگام استفاده از مجموعه هایی که بر پایه کد هشینگ کار می کنند مفید است. به عنوان مثال کلاس `Dictionary<TKey,TValue>` یکی از این موارد است.

یک کلاس به نام `PersonDetail` به صورت زیر تعریف کنید:

```
public class PersonDetail
{
    public string Favorites { get; set; }
}
```

حال فرض کنید بخواهیم یک دیکشنری از `Person` و `PersonDetail` ایجاد کنیم. به این صورت که برای هر `Person` به عنوان کلید یک `PersonDetail` به عنوان مقدار داشته باشیم.

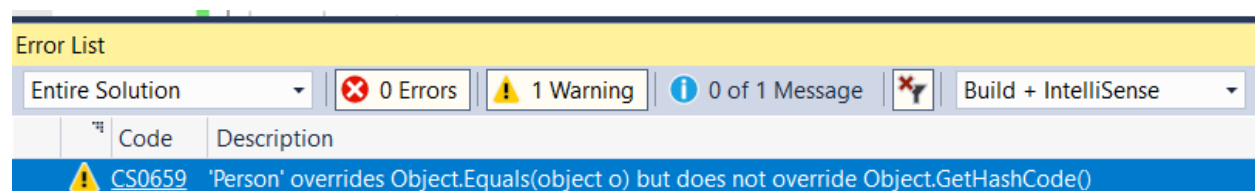
دستورات زیر را در نظر بگیرید :

```
PersonDetail p1d = new PersonDetail { Favorites = "Programming,Traditional Music,..." };
PersonDetail p2d = new PersonDetail { Favorites = "Watching TV,Walking,..." };

Dictionary<Person, PersonDetail> persons = new Dictionary<Person, PersonDetail>
{
    { new Person { Age = 10 }, p1d},
    { new Person { Age = 10}, p2d},
    { new Person { Age = 20 }, p1d},
    { new Person { Age = 20}, p2d}
};
```

دستورات فوق بدون هیچ ایراد نحوی کمپایل و اجرا می شود. اما در دستورات فوق یک ایراد منطقی وجود دارد و آن هم این است که در دیکشنری `Persons` کلید های تکراری وجود دارد. توجه داشته باشید که برابری `Person` ها بر اساس `Age` آن ها می باشد. لذا مدخل های اول و دوم و همچنین مدخل های سوم و چهارم در دیکشنری `persons` یکسان می باشند و از طرفی چون درون یک دیکشنری می بایستی کلید ها یکتا باشند، دستورات فوق دارای ایراد منطقی است. این خطا از آن جهت اتفاق افتاده است که کلاس `Dictionary` برای بررسی یونیک بودن کلید های خود از متد `GetHashCode` استفاده می کند و چون این متد در کلاس `Person` دوباره نویسی نشده و عملکرد پیش فرض خود را دارد، لذا کلید مدخل های اول و دوم و همچنین کلید مدخل های سوم و چهارم متفاوت بوده و بنابر این کلاس `Dictionary` با وجود مقادیر یکسان `Age` برای این مدخل ها، آن ها را متفاوت در نظر می گیرد.

به همین دلیل پیشنهاد می شود هر زمان که متد `Equals` را دوباره نویسی می کنید، متد `GetHashCode` را نیز دوباره نویسی کرده تا بتوانید رفتار آن را کنترل نمایید. این پیشنهاد به صورت یک `warning` نیز در ویژوال استودیو به شما یادآوری می شود.



برای حل این مورد دستورات کلاس `Person` را به صورت زیر تغییر دهید :

```
public class Person
{
    public int Age { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }

    public override bool Equals(object obj)
    {
        if (obj is null)
            return false;
        Person other = obj as Person;
        if (other == null)
            return false;
        return this.Age == other.Age;
    }
    public override int GetHashCode()
    {
        return this.Age.GetHashCode();
    }
}
```

```

    }
    public override string ToString()
    {
        return $"{FirstName},{LastName},{Age}";
    }
}

```

با این تغییرات، کد هشینگ ایجاد شده برای هر شخص وابسته به مقدار Age می باشد.

حال اگر دستورات مربوط به دیکشنری را مجدداً اجرا کنید، خطای زمان اجرای زیر را خواهید داشت :

```

Dictionary<Person, PersonDetail> persons = new Dictionary<Person, PersonDetail>
{
    { new Person { Age = 10 }, p1d},
    { new Person { Age = 10}, p2d},
    { new Person { Age = 20 }, p1d},
    { new Person { Age = 20}, p2d}
};

```

✖

Exception Unhandled
🔍 ✕

Con: **System.ArgumentException:** 'An item with the same key has already been added.'

همانطور که مشاهده می کنید برنامه ایجاد Exception نموده و دیکشنری ورود کلید های تکراری را تشخیص داده است.

نکته : نحوه ایجاد کد های هشینگ نیز خود مبحث مفصلی است که در مقاله ای دیگر بدان اشاره خواهد شد.

جمع بندی

در این مقاله سعی شد کلیات مباحث مربوط به مقایسه انواع، برابری ها و شخصی سازی های موردی برای انواع سفارشی مورد بررسی قرار گیرند. بدیهی است مثال های ارائه شده در این مقاله ساده انتخاب شده تا درک موضوع برای خواننده راحت تر باشد. لازم است تا خواننده محترم پس از مطالعه این مقاله، بار ها و بار ها و بارها مطالب

را بررسی و کد ها را نوشته و تست نماید تا به درک عمقی از مطالب برسد. امیدوارم این مقاله بتواند در کد نویسی بهتر به شما کمک کند. چنانچه موردی در خصوص مطالب این مقاله نیاز به یادآوری دارد حتما با من در تماس باشید.

منابع

- ✓ <https://docs.microsoft.com/en-us/dotnet/api/system.icomparable?redirectedfrom=MSDN&view=netframework-4.7.2>
- ✓ <https://docs.microsoft.com/en-us/dotnet/api/system.collections.icomparer?redirectedfrom=MSDN&view=netframework-4.7.2>
- ✓ <https://docs.microsoft.com/en-us/dotnet/api/system.comparison-1?redirectedfrom=MSDN&view=netframework-4.7.2>
- ✓ <https://docs.microsoft.com/en-us/dotnet/api/system.object.referenceequals?view=netframework-4.7.2>
- ✓ <https://docs.microsoft.com/en-us/dotnet/api/system.object.equals?view=netframework-4.7.2>

- ✓ https://docs.microsoft.com/en-us/dotnet/api/system.object.gethashcode?redirectedfrom=MSDN&view=netframework-4.7.2#System_Object_GetHashCode
- ✓ [https://docs.microsoft.com/en-us/previous-versions/dotnet/netframework-4.0/7h9bszxx\(v=vs.100\)](https://docs.microsoft.com/en-us/previous-versions/dotnet/netframework-4.0/7h9bszxx(v=vs.100))
- ✓ <https://docs.microsoft.com/en-us/dotnet/api/system.string.isinterned?view=netframework-4.7.2>
- ✓ https://docs.microsoft.com/en-us/dotnet/api/system.object.memberwiseclone?redirectedfrom=MSDN&view=netframework-4.7.2#System_Object_MemberwiseClone

پایان.

راه های ارتباط با من

Homepage: <http://mkiani.ir>
Email: mkiani3000@gmail.com
Telegram channel: [@codeway](https://t.me/codeway)

مرداد 1397