

Comparisons in .NET

Simon Robinson
<http://TechieSimon.com>
@TechieSimon



pluralsight 
hardcore dev and IT training

➔ What is a comparison?

- Differences between comparisons and equality.

➔ Support for comparisons:

- `Comparable<T>` and the `CompareTo()` method.
- C# comparison operators: `<`, `<=`, `>`, `>=`.

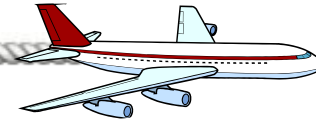
➔ How and why to implement comparisons in your types.

➔ Consuming in generic code:

- `Comparable<T>` works.
- C# comparison operators don't work.

What is a Comparison

Comparison: Way of ordering objects



```
if (3 < 4) {
```



true

because 3 comes before 4

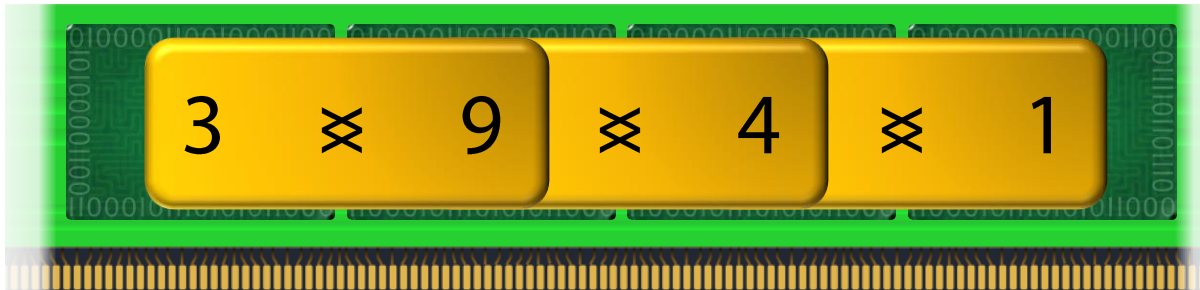


Comparing and Sorting

Sorting: Important application of comparing



If you can compare...
... you can sort!



Collections often need to do this
(Sort their elements)

Comparisons and Equality

If the comparison
says...

Then equality
says..

$x = y$

Equal

$x > y$

$x < y$

Not equal

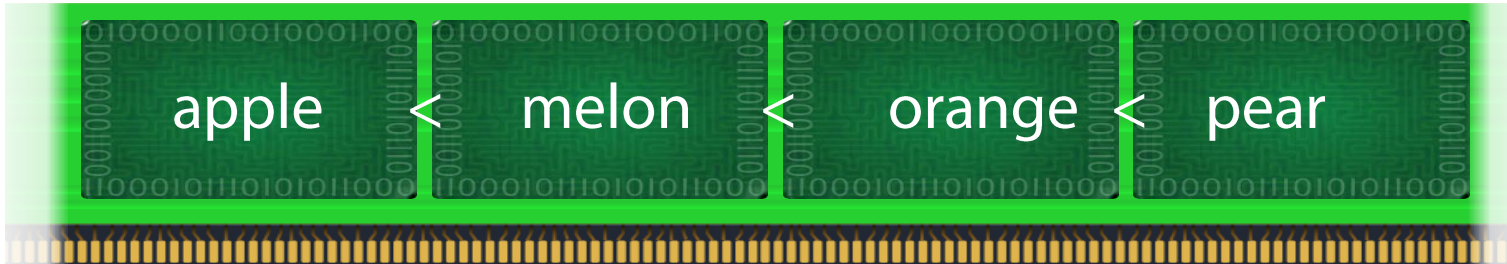


Equality
is a special case of
comparisons

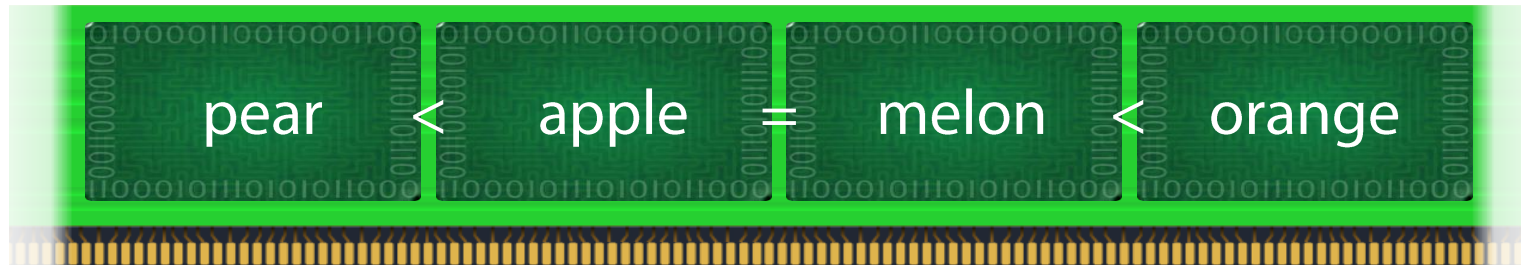
'Natural' and 'Plugged-In' Comparisons

Eg. For strings...

Natural comparison: Alphabetical



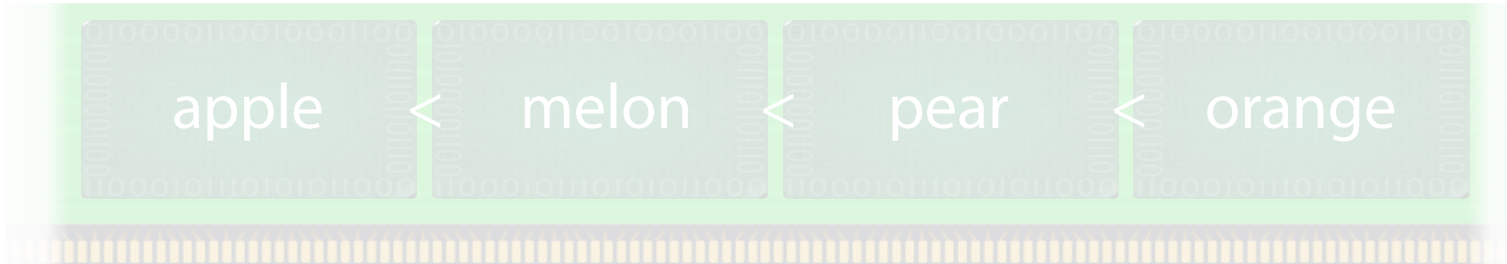
A 'plugged-in' comparison: Compare by string length



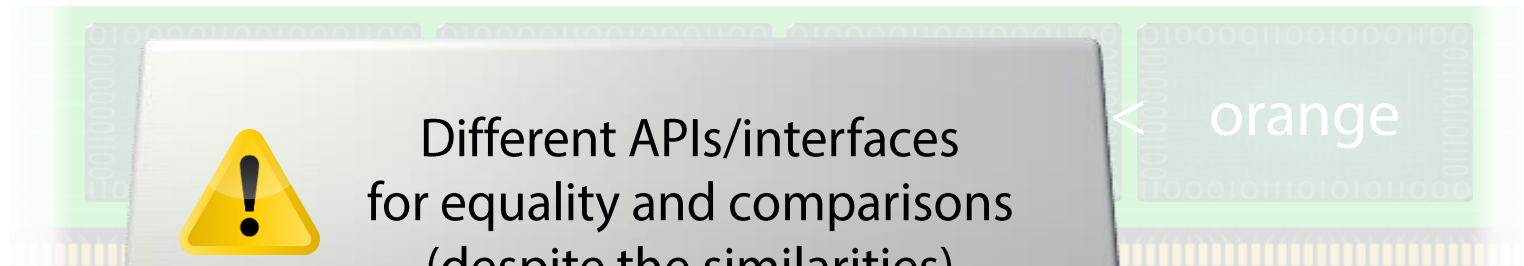
Different APIs

Eg. For strings...

Natural comparison: Alphabetical



A 'plugged-in' comparison: Compare by string length



Different APIs/interfaces
for equality and comparisons
(despite the similarities)

Equality vs Comparisons in .NET

Equality

"Natural"

`object.Equals()`
(and other methods)

`IEquatable<T>`

`==, !=` operators



Comparisons



No support in
`System.object`

`IComparable`
`IComparable<T>`



`>, <, >=, <=` operators

"Plugged-in"

`IEqualityComparer<T>`



`IComparer<T>`



object Doesn't Support Comparisons

Equality

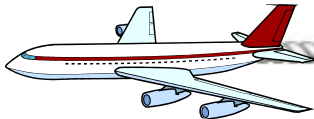
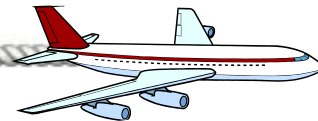
`object.Equals()`
(and other methods)

Comparisons



No support in
`System.object`

Equality makes sense for **all** types



Comparisons **don't** make sense for many
types



Comparisons

Is $3 < 4$?



This question makes sense



Is

OK

<

Continue

?



This question is nonsense



For most types,
comparisons
don't make sense



Equality vs Comparisons in .NET

Equality

`object.Equals()`
(and other methods)

`IEquatable<T>`

`==, !=` operators

Supplement to
`System.object` methods

Comparisons



No support in
`System.object`

`IComparable`
`IComparable<T>`

`>, <, >=, <=` operators

The way that a type declares
it knows how to compare instances

Comparisons are Value Only

Equality: Reference or value

Comparisons: Value only

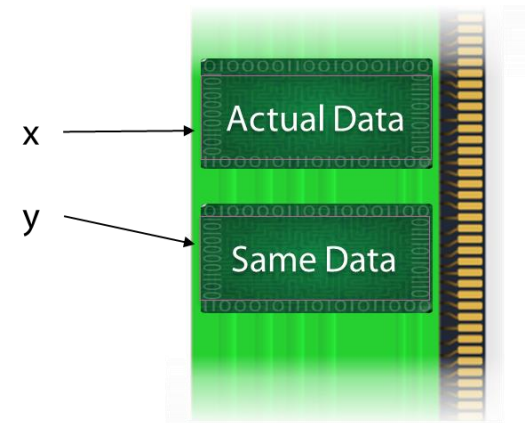
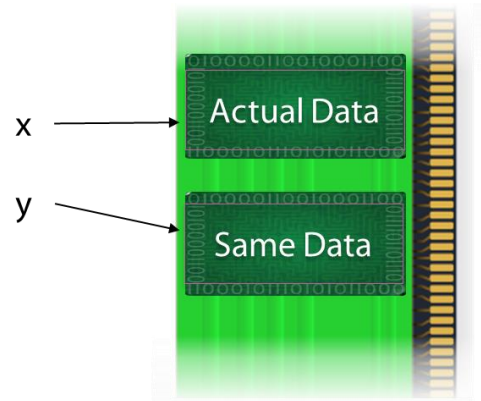
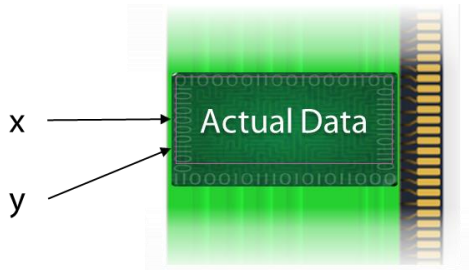
$x == y$

$x > y$

Same address?

Same value?

Compare values



Equality and Comparison Operators

Equality

`object.Equals()`
(and other methods)

`IEquatable<T>`

`==, !=` operators

Work out of the box
for all
primitive and reference types

Comparisons



No support in
`System.object`

`IComparable`
`IComparable<T>`

`>, <, >=, <=` operators

Work out of the box
only for
primitive types.

Implementing Comparisons

More often than not...



Implementing Comparisons

More often than not...



... Don't! ...



Comparing Foods....

Want add ability to sort foods



```
public class Food : IComparable<Food>
{
    public int CompareTo(Food other)
    {
        // what do you put here?
    }
}
```


Comparing Foods....

Compare by name?

Compare by
food group?

Compare by calories?

```
public class Food : IComparable<Food>
{
    public int CompareTo(Food other)
    {
        // what do you put here?
    }
}
```

These are all
good
comparisons

None are
natural
comparisons

Comparing Foods....

Compare by name

```
public class Food : IComparable<Food>
{
    public int CompareTo(Food other)
    {
```



IComparable<T>
is not appropriate for
Food

by name



But writing a comparer
for Food
is fine
a natural comparison



Extra Issues for Reference Types



More checks in strongly typed CompareTo()...

```
// if CalorieCount is a class
public int CompareTo(CalorieCount other)
{
    if (other == null)
        return 1; // any instance comes after null
    if (ReferenceEquals(other, this))
        return 0;
    if (other.GetType() != this.GetType())
        // probably can't handle this
        throw new ArgumentException();
    // the logic - finally
    return this._value.CompareTo(other._value);
}
```

Extra Issues for Reference Types



If CalorieCount is an unsealed class...

... this happens...



```
public int CompareTo(CalorieCount other)
{
```

If this is a derived type instance
– good luck!

I suggest:

Avoid implementing
comparisons
on nonsealed classes

Generic Code

Recall for equality...

```
object.Equals()
```

✓ OK
in generic code

It's the same
for comparisons...

```
IComparable<T>  
.CompareTo()
```

```
operator ==  
operator !=
```

Don't work
in generic code



```
operator >  
operator >=  
etc.
```

Code Demo

**Do Not Place Anything
in This Space**

(Add watermark during
editing)

Note: Warning will not appear
during Slide Show view.

Summary

- ➔ To implement comparisons on your types:
 - Implement `IComparable<T>`.
 - Optionally implement comparison operators.
- ➔ Microsoft types:
 - Most types have no support for comparisons.
 - `int`, `float`, etc. implement `IComparable<T>` and the operators.
 - `string` implements `IComparable<T>` only.
- ➔ Usually, writing a custom comparer is better than implementing comparisons.
- ➔ Consuming comparisons in generic code requires the interface, not the operators.