


# Comparers and Equality Comparers

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



**pluralsight**   
hardcore dev and IT training



→ What comparers and equality comparers are.  
- Why collections rely on them.

→ Comparer demo:  
- Allow sorting `Food` instances.

→ Equality comparer demo:  
- Ignoring case in a hash set of `FoodItem` instances.

→ Default comparers.

→ String Comparers.

# Comparers

A **comparer** knows how to compare other objects (according to some criteria)!





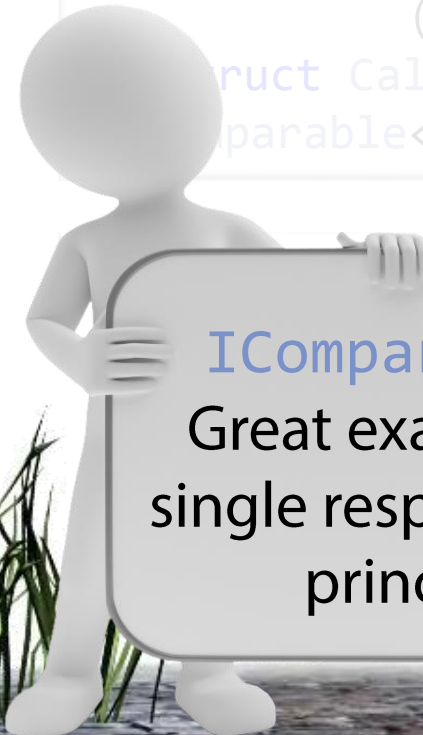
# IComparer<T> vs IComparable<T>

## IComparable<T>

Implemented by T

(eg.

```
struct CalorieCount :  
    IComparable<CalorieCount>)
```



**IComparer<T>:**  
Great example of  
single responsibility  
principle

## IComparer<T>

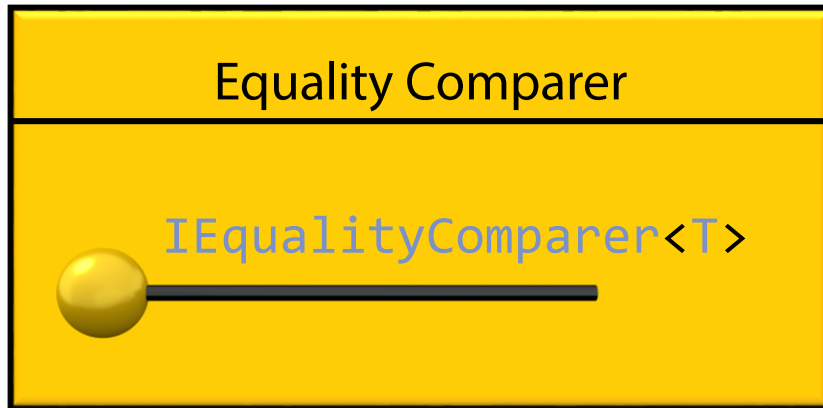
Implemented  
by a custom comparer  
(eg.

```
class FoodNameComparer :  
    IComparer<Food>)
```

Allows 'plugging in' alternative  
comparisons

Means you can have as many  
comparers as you want for T

# Equality Comparers



Like `IComparer<T>`  
but for equality comparisons



# What Are Default Comparers?

Default comparer of **T**

Wrapped in comparer semantics

**Comparer**<T>

**T**'s natural comparisons

**Comparable**<T>

You rarely need  
to deal with  
default comparers  
explicitly

They are always  
instantiated by  
**Comparer**<T>.Default



# Summary

➔ Collections use comparers and equality comparers.

➔ **Comparer:**

- Implements `IComparer<T>`.
- Derive from `Comparer<T>` to implement.

➔ **Equality comparer:**

- Implements `IEqualityComparer<T>`.
- Derive from `EqualityComparer<T>` to implement.

➔ **Default comparers and equality comparers:**

- Give natural equality with comparer semantics.

➔ `StringComparer` – provided by Microsoft.