You can specify the **"OrderBy" criteria dynamically** with the ***IdeaBlade.Linq.SortSelector*** when you can't determine the sort order at compile time.

You create an instance of the *SortSelector* class by specifying the type of object to sort, the property to sort, and its sort direction. The *OrderBySelector* extension method makes it easy to use a *SortSelector* in a standard LINQ query you would otherwise have used *OrderBy*, *OrderByDescending*, *ThenBy* or *ThenByDescending*. For example:

```
var sort = new SortSelector(typeof(Customer), "Country", ListSortDirection.Descending);
var customers = myEntityManager.Customers.OrderBySelector(sort).ToList();
// compare with the strongly-typed LINQ equivalent
var customers = myEntityManager.Customers.OrderByDescending(c => c.Country).ToList();
```

```
Dim sort = New SortSelector(GetType(Customer), "Country", ListSortDirection.Descending)
Dim customers = myEntityManager.Customers.OrderBySelector(sort).ToList()
' compare with the strongly-typed LINQ equivalent
Dim customers = myEntityManager.Customers.OrderByDescending(Function(c) c.Country).ToList()
```

A single *SortSelector* instance can specify multiple sort properties, each with its own sort direction. You add sort criteria with the *ThenBy* method.

```
var sort = new SortSelector(typeof(Customer), "Country", ListSortDirection.Descending);
sort = sort.ThenBy("CompanyName");  // by default ListSortDirection is Ascending if not specified.
var customers = myEntityManager.Customers.OrderBySelector(sort).ToList();
// compare with the strongly-typed LINQ equivalent
var customers = myEntityManager.Customers
  .OrderByDescending(c => c.Country)
  .ThenBy(c => c.CompanyName)
  .ToList();
```

```
Dim sort = New SortSelector(GetType(Customer), "Country", ListSortDirection.Descending)
sort = sort.ThenBy("CompanyName") ' by default ListSortDirection is Ascending if not specified.
Dim customers = myEntityManager.Customers.OrderBySelector(sort).ToList()
' compare with the strongly-typed LINQ equivalent
Dim customers = myEntityManager.Customers
  .OrderByDescending(Function(c) c.Country)
  .ThenBy(Function(c) c.CompanyName)
  .ToList()
```

*SortSelector*s can also be combined with an overload of the *ThenBy* method or with the static *Combine* method.

```
// create two SortSelectors
var sort1 = new SortSelector(typeof(Customer), "Country", ListSortDirection.Descending);
var sort2 = new SortSelector(typeof(Customer), "CompanyName");
// combine them using either of the following two mechanisms.
var sort = sort1.ThenBy(sort2);
// or
var sort = SortSelector.Combine(new[] { sort1, sort2 });
```

```
' create two PropertySelectors
Dim sort1 = New SortSelector(GetType(Customer), _
 "Country", ListSortDirection.Descending)
Dim sort2 = New SortSelector(GetType(Customer), "CompanyName")
' combine them using either of the following two mechanisms.
Dim sort = sort1.ThenBy(sort2)
' or
Dim sort = SortSelector.Combine( { sort1, sort2 })
```

The syntax and behavior of a *SortSelector* are similar to the *PredicateDescription* used in a *Where* clause as described here and here. For example, you can delay specifying the type of entity to sort until you actually use the *SortSelector*.

```
// Type not specified; it is determined when used
var sort = new SortSelector("Country", ListSortDirection.Descending);
var customers = myEntityManager.Customers.OrderBySelector(sort).ToList();
```

```
' Type not specified; it is determined when used
Dim sort = New SortSelector("Country", ListSortDirection.Descending)
Dim customers = myEntityManager.Customers.OrderBySelector(sort).ToList()
```

Such lazy-typing can be useful when sorting anonymous types.