CarLister: Homework #3 – Add list sorting

Task 1) Add action links to the column headings in the list view

1. Open the car list view (**Views/Car/Index.cshtml**) and change the column heading layout as follows (see the foreach loop added below the commented line):  
     
   <tr>  
    <th colspan="6">  
    Check out my cars!  
    </th>  
   </tr>  
   <tr style="background-color:gainsboro">  
    @\*Added to accommodate sorting by column heading - to be added later\*@   
    @foreach (var col in new[] { "Year", "Make", "Model", "Price" })  
    {  
    <th>@Html.ActionLink(col, "Sort", new { property = col })</th>  
    }  
    <th>  
    Picture  
    <div style="font-size:15px">  
    (Hover for larger image<br />Click for full size)  
    </div>  
    </th>  
    <th></th>  
   </tr>

Task 2) Write a Sort method that will be called whenever a column heading is clicked

1. The action link applied to each column heading references a **Sort** action, which means we need to build a **Sort method** in the **Cars controller**. We’ll also need a couple of special attributes to help us keep track of our current sorting order. Open the Cars controller and add the following attributes to the **CarsController** class:  
     
   static bool SortDirection;  
   static string SortProperty;  
     
   These attributes are declared as static. This means they will retain their values throughout the user’s session. Important for sorting.
2. Now add a new **Sort** method with the following definition and a declared instance of our view model for the list view:  
     
   public ActionResult Sort(string property)  
   {  
    IEnumerable<CarViewModel> model;  
     
   }
3. Next, in this method we need to check the value of the current sort column (SortProperty) and compare it with the newly selected one (property). If they are equal, change the SortDirection. If they are not, simply set a default SortDirection (false – this means we will sort in ascending order). Then set the static SortProperty attribute equal to the newly selected property.  
     
   if (SortProperty == property)  
   {  
    // toggle direction  
    SortDirection = !SortDirection;  
   }  
   else  
   {  
    // initial direction (ascending)  
    SortDirection = false;  
   }  
   SortProperty = property;
4. Next, set up a **switch** statement on the **property** parameter that is passed to the Sort method. Set up cases for each of the sortable values (Year, Make, Model, Price). Each case must check the SortDirection. If false, get a new list in ascending order by that attribute. If true, get a new list in descending order by that attribute. For example:  
     
   case "Year":  
    if (!SortDirection)  
    {  
    model = db.Cars.ToList().OrderBy(car => car.Year);  
    }  
    else  
    {  
    model = db.Cars.ToList().OrderByDescending(car => car.Year);  
    }  
    break;
5. Include a default: case that orders the model by **Id**.
6. MVC has a TempData dictionary object that can be used to share data between controller actions. This is key for us. We need to share the model we’ve just created with our list controller, Index(). The value of TempData persists until it is read or until the current user’s session times out. Persisting data in TempData is useful in scenarios such as redirection, when values are needed beyond a single request.   
     
   Create a “model” index of this dictionary and assign it to your newly ordered model:  
     
   TempData["model"] = model;
7. Finally, **redirect** control to the **Index()** controller.

Task 3) Modify the list controller, Index(), to look for an updated model

1. The **Index()** method should include an instance of our list model (IEnumerable<CarViewModel> model). If it is not present, add it at the beginning of the method.
2. Check the contents of the **TempData** dictionary for a valid model, that is, if it is not null, we have a sorted list coming through. If this is the case, assign the value of the TempData dictionary’s “model” index to the instance of our model for this method.  
     
   if (TempData["model"] != null)  
   {  
    model = (IEnumerable<CarViewModel>)TempData["model"];  
   }
3. If the value at the dictionary’s “model” index is null, set the model to a list ordered by **Id** and return the view, passing it our model.
4. Build your program, correct any errors and test it for correct operation. You should be able to click on the column headings to sort your rows, toggling between ascending and descending order.