Session 14 – Comparators

Lab 01 - Demo string and int IComparable.CompareTo() implementation

1. Create a new Console Project and add the following code to the Program.Main()

```
class Program
{
    static void Main(string[] args)
    {
        string apple = "apple";
        string pear = "pear";

        DisplayOrder(apple, pear);
        DisplayOrder(pear, apple);
        DisplayOrder(apple, apple);

        DisplayOrder(3, 4);
        DisplayOrder(4, 3);
        DisplayOrder(3, 3);
    }

    static void DisplayOrder<T>(T x, T y) where T : IComparable<T>
    {
        int result = x.CompareTo(y);
        if (result == 0)
            Console.WriteLine("{0,12} = {1}", x, y);
        else if (result > 0)
            Console.WriteLine("{0,12} > {1}", x, y);
        else
            Console.WriteLine("{0,12} < {1}", x, y);
    }
}
```

2. Add another string called “Apple” and compare it with an “apple’ and a “pear” – What did you observe?
3. How does the string and int types know their natural ordering?
4. Try using the comparator operators such as >, >=, <, <=.
5. What did you observe and why?

Lab 02 – Example of using IComparable of a Reference type

1. Open a new Console project and create a new class called CalorieCount that implements IComparable<CalorieCount>
2. Copy and paste the code found in the Lab02.txt file found in session14 resources folder, to implement the IComparable.CompareTo()
3. Open the Program.Main() and add the code below:

```
static void Main(string[] args)
{
    CalorieCount cal300 = new CalorieCount(300);
    CalorieCount cal400 = new CalorieCount(400);

    DisplayOrder(cal300, cal400);
    DisplayOrder(cal400, cal300);
    DisplayOrder(cal300, cal300);

    if (cal300 < cal400)
        Console.WriteLine("cal300 < cal400");

    if (cal300 == cal400)
        Console.WriteLine("cal300 == cal400");

}


static void DisplayOrder<T>(T x, T y) where T : IComparable<T>
{
    int result = x.CompareTo(y);
    if (result == 0)
        Console.WriteLine("{0,12} = {1}", x, y);
    else if (result > 0)
        Console.WriteLine("{0,12} > {1}", x, y);
    else
        Console.WriteLine("{0,12} < {1}", x, y);
}
```

4. Run the Main() – What did you observe?

Lab 03 – Using with Comparers with ICompare

1. Create a new Console project.
2. Add the Food class you created in session 13 labs last week.
3. Open the Main() and add the following code

```
class Program
{
    static void Main(string[] args)
    {
        Food[] list = {
            new Food("orange", FoodGroup.Fruit),
            new Food("banana", FoodGroup.Fruit),
            new Food("pear", FoodGroup.Fruit),
            new Food("apple", FoodGroup.Fruit) };

        Array.Sort(list);

        foreach (var item in list)
            Console.WriteLine(item);
    }
}
```

4. Run the code. What did you observe and why
5. Add a new class called FoodNameComparer and add the following code:

```
class FoodNameComparer : Comparer<Food>
{
    private static FoodNameComparer _instance = new FoodNameComparer();

    public static FoodNameComparer Instance { get { return _instance; } }

    private FoodNameComparer() { }

    public override int Compare(Food x, Food y)
    {
        if (x == null && y == null)
            return 0;
        if (x == null)
            return -1;
        if (y == null)
            return 1;
        return string.Compare(x.Name, y.Name, StringComparison.CurrentCulture);
    }
}
```

6. Open the Main() and make the following highlighted change:

```
static void Main(string[] args)
{
    Food[] list = {
        new Food("orange", FoodGroup.Fruit),
        new Food("banana", FoodGroup.Fruit),
        new Food("pear", FoodGroup.Fruit),
        new Food("apple", FoodGroup.Fruit) };

    Array.Sort(list, FoodNameComparer.Instance);

    foreach (var item in list)
        Console.WriteLine(item);
}
```

7. Run from the Main() – Why did you obderve? Can you explain the change?

Lab 04 – Comparators and Sub classes

1. Create a new Console project
2. To this project, add the Food CookedFood classes you created in session 13 last week
3. Add the FoodNameComparer class you created in the previous project here.
4. Open the Main() and add the following test code:

```
static void Main(string[] args)
{
    // lists will be sorted differently
    //because the comparer is unable to distinguish
    // apple and baked apple
    Food[] list = {
        new Food("apple", FoodGroup.Fruit),
        new Food("pear", FoodGroup.Fruit),
        new CookedFood("baked", "apple", FoodGroup.Fruit),
     };
    SortAndShowList(list);

    Food[] list2 = {
        new CookedFood("baked", "apple", FoodGroup.Fruit),
        new Food("pear", FoodGroup.Fruit),
        new Food("apple", FoodGroup.Fruit),
     };
    Console.WriteLine();
    SortAndShowList(list2);

}

static void SortAndShowList(Food[] list)
{
    Array.Sort(list, FoodNameComparer.Instance);

    foreach (var item in list)
        Console.WriteLine(item);
}
}
```

5.Run the Main(). What did you observe? – Can you explain it?

Lab 05 – Using string class Comparers

1. Open a new Console project and add the following code to the Main()

```
static void Main(string[] args)
{
    var names = new HashSet<string>(StringComparer.CurrentCulture);
    names.Add("apple");
    names.Add("pear");
    names.Add("pineapple");
    names.Add("apple");

    foreach (string name in names)
        Console.WriteLine(name);
}
```

2. Run the Main() , what did you observe? – why?
3. Make the following changes to the Main():

```
class Program
{
    static void Main(string[] args)
    {
        var names = new HashSet<string>(StringComparer.CurrentCultureIgnoreCase);
        names.Add("apple");
        names.Add("pear");
        names.Add("pineapple");
        names.Add("Apple");

        foreach (string name in names)
            Console.WriteLine(name);
    }
}
```

4.Run the Main(), what did you now observe- Why?

Lab 0x – Using the Equality Comparers

1. Open a new Console project
2. Copy the Food class you created in the previous session labs to this project
3. Create a new class called FoodItemEqualityComparer that implements the EqualityComparer<FoodItem>
4. Add the following code to this class as shown below:

```
class FoodItemEqualityComparer : EqualityComparer<FoodItem>
{
    private static readonly FoodItemEqualityComparer _instance =
        new FoodItemEqualityComparer();
    public static FoodItemEqualityComparer Instance { get { return _instance; } }
    private FoodItemEqualityComparer() { }

    public override bool Equals(FoodItem x, FoodItem y)
    {
        throw new NotImplementedException();
    }

    public override int GetHashCode(FoodItem obj)
    {
        throw new NotImplementedException();
    }
}
```

5. Add the following code to the Program.Main() to test the class for Equality:

```
class Program
{
    static void Main(string[] args)
    {
        var foodItems = new HashSet<FoodItem>(FoodItemEqualityComparer.Instance);
        foodItems.Add(new FoodItem("apple", FoodGroup.Fruit));
        foodItems.Add(new FoodItem("pear", FoodGroup.Fruit));
        foodItems.Add(new FoodItem("pineapple", FoodGroup.Fruit));
        foodItems.Add(new FoodItem("Apple", FoodGroup.Fruit));

        foreach (var foodItem in foodItems)
            Console.WriteLine(foodItem);
    }
}
```

6. Run the program. What did you observe? How can to ensure that only one instance of Apple can be added?
7. Implement the two methods of the EqualityComparer<FoodItem>as shown below:

```
public override bool Equals(FoodItem x, FoodItem y)
{
    return x.Name.ToUpperInvariant() == y.Name.ToUpperInvariant()
        && x.Group == y.Group;
}

public override int GetHashCode(FoodItem obj)
{
    return obj.Name.ToUpperInvariant().GetHashCode() ^
        obj.Group.GetHashCode();
}
```

8. Now run the Main() – What was different and why?