Session 13 – Equality and Comparisons Lab Exercises

Lab 01 – int and Integers Equality

1. Open VS 2015 and create a Console project in C# .
2. Declare two int variables and compare them for equality using the C# '==' operator and .NET Equals( ) method. Did both comparisons yield the same results?
3. Modify the above code to use Integer objects instead of int primitive type.
4. Test the two objects for equality using the '==' and Equals( ). What were the results?

Lab 02- Floating point tests for Equality

1. Create a new Console project in C#
2. Declare two floating point numbers and test them for equality using the '==' and Equals( ) . What

Lab 03 – Testing Strings for Equality

1. Create a new Console project in C#
2. Prompt the user to enter two string values using the keyboard
3. Test the two string for equality using the '==' and the Equaks()
4. Using different sets of values to test for equality.

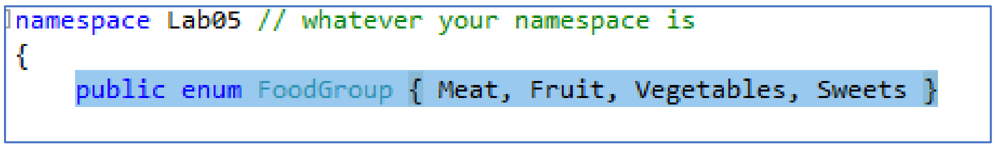
Lab 04 – Testing Reference Types for Equality.

1. Create a new Console project in C#
2. Create a class called Food with a Name property of type string.
3. Create two instances of the Food class.
4. Test the two food instances for Equality using the '==' operator and Equals()
5. Did both operators give you the same or different result for two different food instances with the same Name property – Can you explain the results

Lab 05 – Overloading the '==' operator and Overriding the Equals() on a Reference type

1. Using the class Food you created in Lab 04 make the following changes

2. Add an enumeration of Food Groups as shown below



```
namespace Lab05 // whatever your namespace is
{
    public enum FoodGroup { Meat, Fruit, Vegetables, Sweets }
```

3. Next Add a FoodGroup type property as shown below

```
public class Food
{
    public String Name { get; set; }
    public FoodGroup Group { get; set; }
```

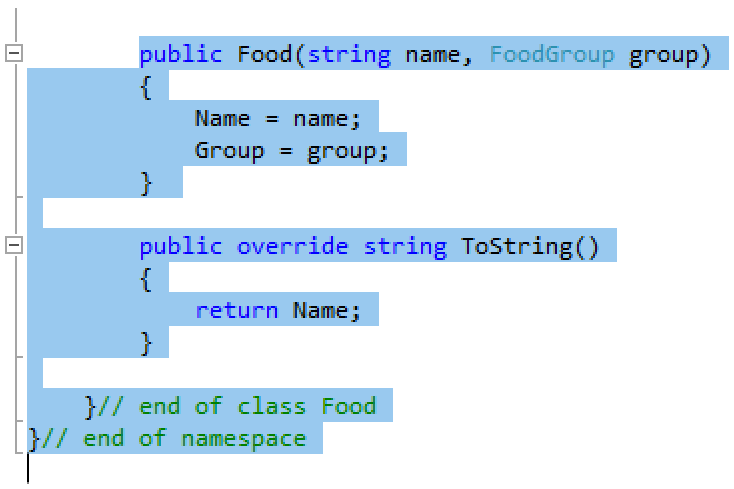
4. Next, overload the '==' and '!=' operators as follows:

```
public class Food
{
    public String Name { get; set; }
    public FoodGroup Group { get; set; }

    public static bool operator ==(Food x, Food y)
    {
        return object.Equals(x, y);
    }

    public static bool operator !=(Food x, Food y)
    {
        return !object.Equals(x, y);
    }
```

5. Next, override the Equals( ) and GetHashCode( ) as shown below

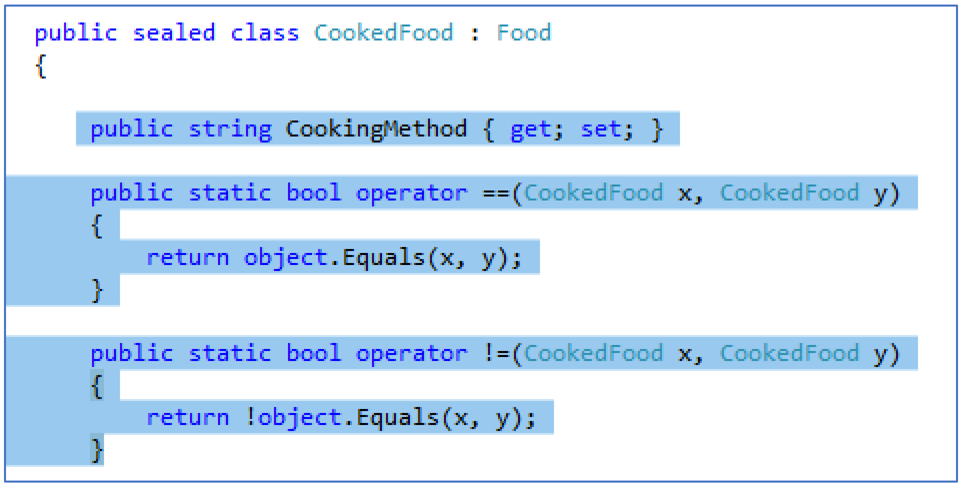
```
public override bool Equals(object obj)
{
    if (obj == null)
        return false;
    if (ReferenceEquals(obj, this))
        return true;
    if (obj.GetType() != this.GetType())
        return false;
    Food rhs = obj as Food;
    return this.Name == rhs.Name && this.Group == rhs.Group;
}

public override int GetHashCode()
{
    return this.Name.GetHashCode() ^ this.Group.GetHashCode();
}
```

6. Next add a Food constructor and override the ToString() as follows:



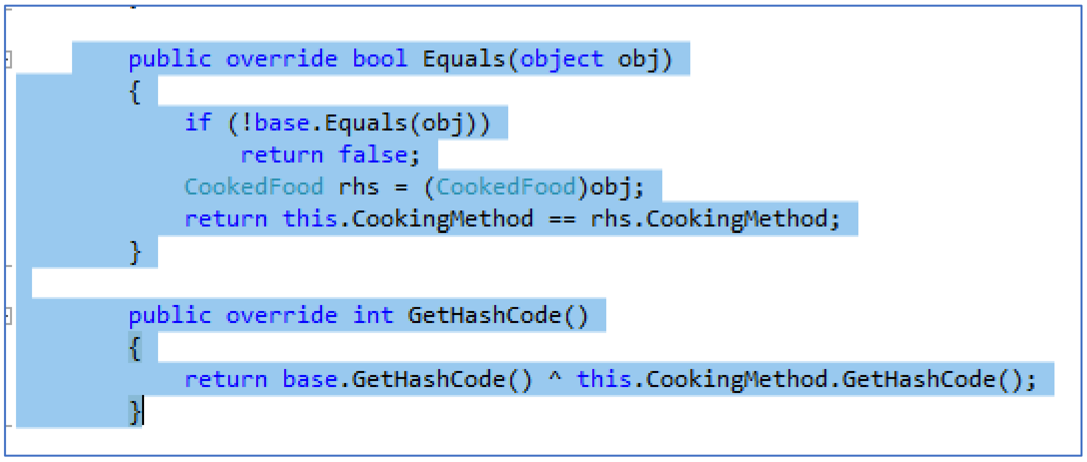
```
        public Food(string name, FoodGroup group)
        {
            Name = name;
            Group = group;
        }

        public override string ToString()
        {
            return Name;
        }

    }// end of class Food
}// end of namespace
```

7. Next, create a sealed subclass CookedFood that inherits from Food class and add a string property CookingMethod and overload the '==' and '!=' operators as follows:



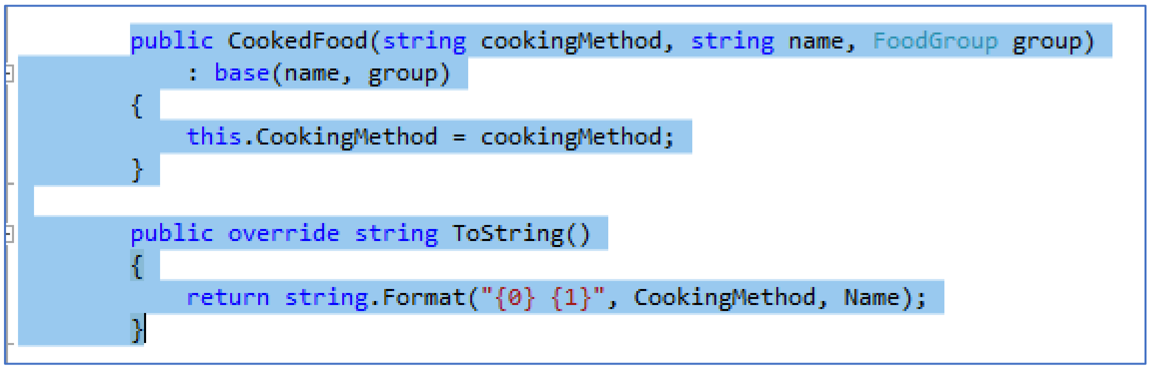
```
public sealed class CookedFood : Food
{

    public string CookingMethod { get; set; }

        public static bool operator ==(CookedFood x, CookedFood y)
        {
            return object.Equals(x, y);
        }

        public static bool operator !=(CookedFood x, CookedFood y)
        {
            return !object.Equals(x, y);
        }
```

8. Next override the Equals( ) and GetHAshCode() as shown below:



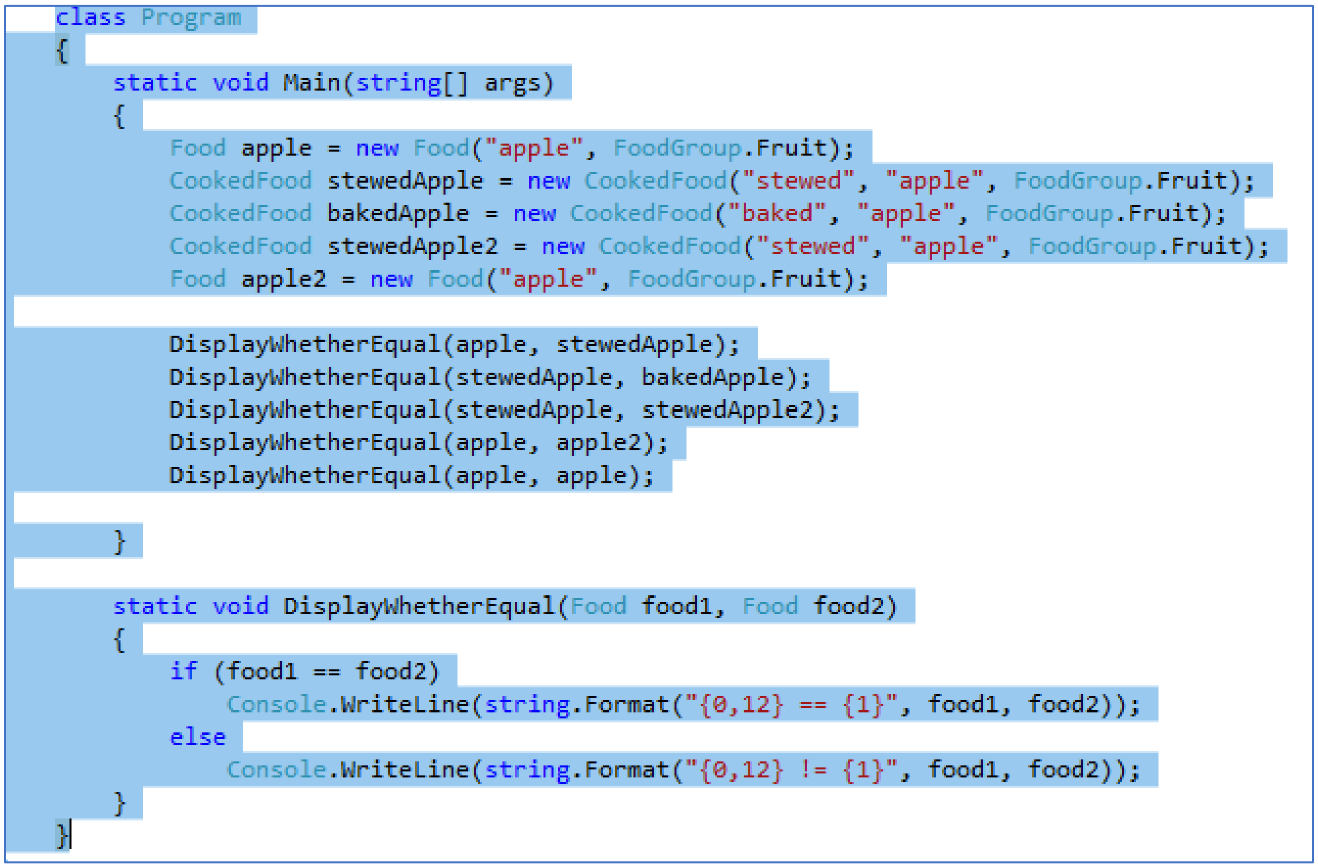
```
        public override bool Equals(object obj)
        {
            if (!base.Equals(obj))
                return false;
            CookedFood rhs = (CookedFood)obj;
            return this.CookingMethod == rhs.CookingMethod;
        }

        public override int GetHashCode()
        {
            return base.GetHashCode() ^ this.CookingMethod.GetHashCode();
        }
```

9. Next, create an all args Constructor and override the ToString() as follows:



```
        public CookedFood(string cookingMethod, string name, FoodGroup group)
            : base(name, group)
        {
            this.CookingMethod = cookingMethod;
        }

        public override string ToString()
        {
            return string.Format("{0} {1}", CookingMethod, Name);
        }
```

10. Open the Program class Main(..) and enter the following code to test the classes for Equality:



```
    class Program
    {
        static void Main(string[] args)
        {
            Food apple = new Food("apple", FoodGroup.Fruit);
            CookedFood stewedApple = new CookedFood("stewed", "apple", FoodGroup.Fruit);
            CookedFood bakedApple = new CookedFood("baked", "apple", FoodGroup.Fruit);
            CookedFood stewedApple2 = new CookedFood("stewed", "apple", FoodGroup.Fruit);
            Food apple2 = new Food("apple", FoodGroup.Fruit);

            DisplayWhetherEqual(apple, stewedApple);
            DisplayWhetherEqual(stewedApple, bakedApple);
            DisplayWhetherEqual(stewedApple, stewedApple2);
            DisplayWhetherEqual(apple, apple2);
            DisplayWhetherEqual(apple, apple);

        }

        static void DisplayWhetherEqual(Food food1, Food food2)
        {
            if (food1 == food2)
                Console.WriteLine(string.Format("{0,12} == {1}", food1, food2));
            else
                Console.WriteLine(string.Format("{0,12} != {1}", food1, food2));
        }
    }
```

11. Now run the Program class and observe the output. Can you explain the results?

Lab 06 – This Exercise is an Assessable Exercise and needs to be completed and uploaded as part of the course assessment requirements. The lab specification is found in a separate document in Session 13 on Learn.