1 Kursusgang 3 - Interfaces

1. *Consider the given class Car. Implement the IComparable interface on Car, to sort the list of cars by Price.*

   Solution:

```
1  using System.Collections.Generic;
2
3  class Program
4  {
5      public static void Main(string[] args)
6      {
7          List<Car> cars =  new List<Car>()
8          {
9              new Car(){Make="Skoda", Model = "Fabia", Price = 50000},
10             new Car(){Make="Skoda", Model = "Octavia", Price = 60000},
11             new Car(){Make="Fiat", Model = "500", Price = 12345},
12             new Car(){Make="Ford", Model = "Mustang", Price = 9000000},
13             new Car(){Make="Ford", Model = "Mustang", Price = 9000001}
14         };
15         cars.Sort();
16         Console.WriteLine("Sorted by price");
17         foreach(Car car in cars)
18         {
19             Console.WriteLine($"  {car.Make} {car.Model} {car.Price}");
20         }
21     }
22 }
23 class Car : IComparable
24     {
25         public string Make { get; set; }
26         public string Model {get; set; }
27         public int Price { get; set; }
28
29         public int CompareTo(object x)
30         {
31             return Price.CompareTo(((Car) x).Price);
32         }
33     }
```

Listing 1: Comparable interface example

- Inside our Main() function, we start out by making a few cars from line 9 to line 13

- We then implement the IComparable interface to the Car class on line 23
- Because of the new interface, we have to implement a function from that interface - the CompareTo function on line 29 to line 32.
- This function takes any object, so we have to typecast it to a car.
- Now that the Car class can compare on prices, we can sort the cars in our Main() function on line 15
- Then, all the cars and their price are printed out in correct order on line 17 to line 20.

2. *Implement the IComparer<Car> to sort cars by Make, Model and lastly by Price.*

   Solution:

```
1  class CarComparer : IComparer<Car>
2  {
3      public int Compare(Car x, Car y)
4      {
5          int make_compare = x.Make.CompareTo(y.Make);
6          if(make_compare != 0)
7          {
8              return make_compare;
9          }
10
11         int model_compare = x.Model.CompareTo(y.Model);
12         if(model_compare != 0)
13         {
14             return model_compare;
15         }
16         return x.Price.CompareTo(y.Price);
17     }
18 }
```

Listing 2: Comparer interface example

- The new class CarComparer is needed to implement the interface IComparer on the Car type.
- The IComparer requires one method - Compare. This takes two arguments of any object type (in this case, cars). This is implemented on line 3.

- When CompareTo compares two things, it will return a statuscode of either -1, 0 or 1. If the status code is 0, the two things are identical. In this case, we want to sort on Make first, then Model, then Price.
- On line 5, we start out by making a comparison. If this comparison is then zero, we can just immediately sort these two objects on line 8.
- This pattern is repeated until all properties are properly sorted

3. *Implement the interface IComparer<Car> to sort cars by Make, Model and then Price in reverse order.*

   Solution:

   - If you put a - in front of the x in line 16, the Price will be sorted in reverse order.

4. *The interface ITaxable. Program an interface ITaxable with a parameterless operation TaxValue and implement this on the class House and class Bus.*

   Solution:

```
1  public interface ITaxable
2  {
3      decimal TaxValue{get;}
4  }
5
6  public class Bus: Vehicle, ITaxable
7  {
8      public Bus(int numberOfSeats, int regNumber, decimal value) : base(
           regNumber, 80, value)
9      {
10         this.numberOfSeats = numberOfSeats;
11     }
12     public decimal TaxValue => (value / 10) + 105M * numberOfSeats;
13 }
14
15 public class House: FixedProperty, ITaxable
16 {
17     public House(string location, bool inCity, double area, decimal value) :
           base(location, inCity, value)
18     {
19         this.area = area;
20     }
21     public decimal TaxValue {
22         get{
23             if (inCity)
24                 return (estimatedValue / 1000.0M) * 5M + 5M * (decimal)area;
25             else
26                 return (estimatedValue / 1000.0M) * 3M;
27         }
28     }
29 }
```

Listing 3: ITaxable interface example

- The interface is declared on line 1 and has a required method for any implementations on line 3.
- On line 12, the Bus class implements ITaxable.
- On line 21, the House class implements the ITaxable interface.

5. *Demonstrate that taxable house objects and taxable bus objects can be used together as objects of type ITaxable.*

   Solution:

```
1  public static void Main(string[] args)
2  {
3      House h1 = new House("Aarhus", true, 3.3, 200000, 200);
4      House h2 = new House("Aalborg", false, 7.8, 500000, 500);
5      Bus b1 = new Bus(5, 090807, 12345678, 123);
6      Bus b2 = new Bus(6, 010203, 87654321, 876);
7
8      ITaxable[] taxables = {h1, h2, b1, b2};
9
10     foreach(ITaxable taxed in taxables)
11     {
12         Console.WriteLine("{0} {1}", taxed, taxed.TaxValue());
13     }
14 }
```

Listing 4: ITaxable demonstration example

- To demonstrate this, we start by adding a few example houses and busses on line 3 - line 6.
- We then make an array of items with the ITaxable interface on line 8.
- These four items in the array are then printed out along with their taxValue property on line 10 - line 13.

6. *Restructure the GameObject program such that class Die and class Card both inherit an abstract class GameObject. You should write the class GameObject*

   Solution:

```
1  abstract class GameObject
2  {
3      public abstract int GameValue{get;}
4      public abstract GameObjectMedium Medium{get;}
5  }
6
7  class Die : GameObject
8  {
9      public override int GameValue{get {return numberOfEyes;}}
10     public override GameObjectMedium Medium
11     {
12         get{ return GameObjectMedium.Plastic; }
13     }
14 }
15
16 class Card : GameObject
17 {
18     public override int GameValue{get {return numberOfEyes;}}
19     public override GameObjectMedium Medium
20         => GameObjectMedium.Plastic; // same as above
21 }
```

Listing 5: Abstract class vs interface example

- We replace the interface with an abstract class called GameObject. This class has two properties on line 3 and line 4 - GameValue and GameObjectMedium. We mark these properties abstract - since we do not have a default implementation
- The class Die overrides it's inherited properties on line 9 and line 10.
- The class Card overrides it's inherited properties on line 18 and line 19.