Interfaces

C# “Multiple” Inheritance

Contents

- What is an interface?
- Built in interfaces
- Creating your own
- Summary
- References

What is an interface?

- C#, unlike C++, does not support multiple inheritance
- Instead, it provides support for Interfaces
- These are very similar to virtual classes in C++
  - You can use these to group related functionalities, but it is up to the class implementing the interface to implement the functionality itself

What is an interface?

- Classes can implement multiple interfaces at once in addition to one parent class

- Note that if a class implements an interface, the functions of that interface MUST be implemented
  - Failing to do so will result in a compilation error

Inheritance vs Interface

- In general, an interface is a can do relationship, while inheritance is an is-a relationship
  - Example: A car and bus are quite similar to how they transport people, so they could probably derive from a common class. A plane can also transport people, however it transports them differently so it may use a common interface, ITransportable (which Car and Bus would use), but have a different parent class.

- Interfaces are ideal to create mix-ins or enhancements to your classes on a case-by-case basis

- However, interfaces can be hard to modify. If you modify an interface, all classes which implement that interface must now conform to the updated interface
  - This can result in a high maintenance cost

Built in interfaces

- .NET Provides us with many different built in interfaces that have specific purposes
  - IComparable
  - ICloneable
  - IDisposable
  - IFormattable
  - IEnumerable

IComparable

- IComparable allows us to create custom sorting logic for our user-defined classes
- It exposes a single method, CompareTo() which compares two objects against each other
- By implementing this interface, it allows you to call the sorting functions on .NET Containers on your custom classes!

IComparable Example

```
public class Person : IComparable<Person>
{
    public string name;
    public int age;

    public Person(string name, int age)
    {
        this.name = name;
        this.age = age;
    }

    public int CompareTo(Person person)
    {
        Person other = obj as Person;

        return age.CompareTo(other.age);
    }
}
```

```
public void PersonSortExample()
{
    List<Person> people = new List<Person>();

    people.Add(new Person("Bob", 28));
    people.Add(new Person("Jones", 25));
    people.Add(new Person("Mark", 40));
    people.Add(new Person("Jill", 30));

    Console.WriteLine("Before sorting");

    foreach (Person p in people)
    {
        Console.WriteLine(p.name + ": " + p.age);
    }

    Console.WriteLine("After sorting");

    people.Sort();

    foreach (Person p in people)
    {
        Console.WriteLine(p.name + ": " + p.age);
    }
}
```

Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

IDisposable

- Used to release unmanaged resources when the Garbage Collector cleans up memory

- For more implementation details, refer to the Memory Management in C# topic

Creating your own interface

- Creating your own interfaces is quite straightforward:

```
public interface ILoggable
   {
       void Log(); //Output debug information

       string Name
       {
           get;
           set;
       }
   }
```

- It's similar to creating a class but with empty functions and no member variables. However, it can contain properties.

Example

-

```
public class Square : ILoggable
   {
        string name;

        public string Name
        {
            get
            {
                return name;
            }

            set
            {
                name = value;
            }
        }

        public void Log()
        {
            Console.WriteLine("My name is " + name + " and I am a
square!");
        }
   }
```

```
public void InterfaceExample()
{
    Square square = new Square();
    square.Name = "Mr Square";

    square.Log();
}
```

Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Summary

- Interfaces are an excellent way to add functionality to classes in your program

- An interface ensures that implementing classes will always conform to a known layout
  - Be careful! If you have an interface shared by many classes, you will need to update them all should the interface itself change

References

- Microsoft, 2014, *Interfaces*
  - https://msdn.microsoft.com/en-us/library/87d83y5b.aspx

- Chetan Kudalkar, 2007, *Interfaces in C# (For Beginners)*
  - *http://www.codeproject.com/Articles/18743/Interfaces-in-C-For-Beginners*