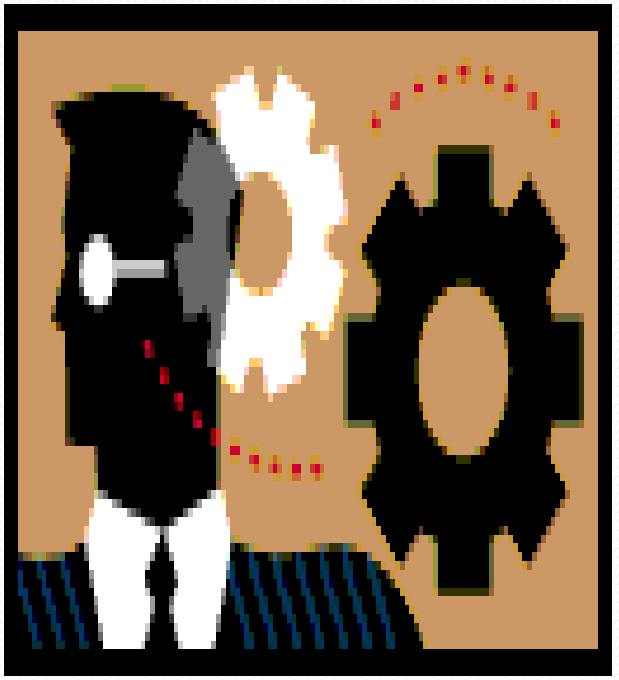
INTERFACES



INTERFACE: THE VEHICLE OF MULTIPLE INHERITANCE

WHAT ARE INTERFACES??

Interfaces are ***reference types***. It is basically class with the following differences:

1. All members of an interface are explicitly PUBLIC and ABSTRACT.

2. Interface cant contain CONSTANT fields, CONSTRUCTORS and DESTRUCTORS.

3. Its members can't be declared STATIC.

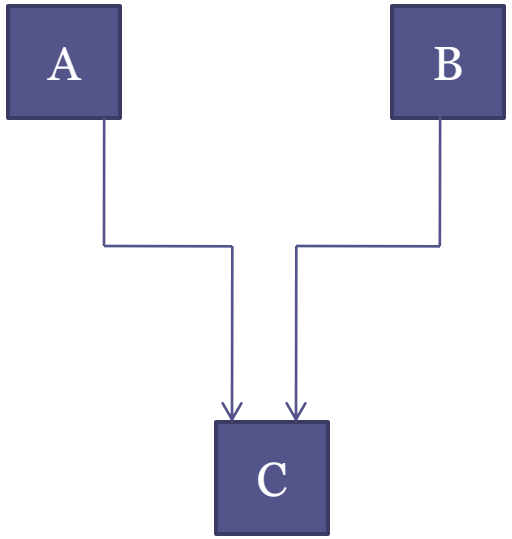
4. All the methods are ABSTRACT, hence don't include any implementation code.

5. An Interface can inherit multiple Interface

WHY INTERFACES???

C# doesn't support MULTIPLE INHERITANCE.

THIS IS POSSIBLE IN C#



THIS IS NOT POSSIBLE IN C#

As evident from the previous diagram, in C#, a class cannot have more than one SUPERCLASS. For example, a definition like:

```
Class A: B,C
{
        .....
        .....
}
```

is *not permitted* in C#.

However, the designers of the language couldn't overlook the importance of the concept of multiple inheritance.
A large number of real life situations require the use of multiple inheritance whereby we inherit the properties of more than one distinct classes.
For this reason, C# provides the concept of interfaces. Although a class can't inherit multiple classes, it can **IMPLEMENT** multiple interfaces.

INTERFACES ALLOW US TO CREATE CLASSES THAT BUILD UPON OTHER CLASSES WITHOUT THE CONFUSION CREATED BY MULTIPLE INHERITANCE.

DEFINING AN INTERFACE

An interface can contain one or multiple *METHODS*, *POPERTIES*, *INDEXERS* and *EVENTS*.
But, these are **NOT** implemented in the interface itself.
They are defined in the class that implements the interface.

Syntax for defining an interface:

interface *interfacename*
{
  member declarations..
}

interface is a KEYWORD.
interfacename is a valid C# identifier.

Example:

```
interface Show
{
  void Display();                    //  method within interface
  int Aproperty
   {
     get;                            // property  within interface
   }
 event someEvent Changed;
void Display();                      // event within interface
}
```

EXTENDING AN INTERFACE

Interfaces can be extended like classes. One interface can be sub interfaced from other interfaces.

Example:

```
interface addition
{
   int add(int x, int y);
}

interface Compute: addition
{
  int sub(int x, int y);
}
```

the above code will place both the methods, add() and sub() in the interface Compute. Any class implementing Compute will be able to implement both these methods.

INTERFACES CAN EXTEND ONLY INTERFACES, THEY CAN'T EXTEND CLASSES. This would violate the rule that interfaces can have only abstract members.

IMPLEMENTING INTERFACES

An Interface can be implemented as follows:

```
class classname: interfacename
{
  body of classname
}
```

A class can derive from a single class and implement as many interfaces required:

```
class classname: superclass, interface1,interface 2,…
{
 body of class name
}
```

eg:

```
class A: B,I1,I2,..
 {
.......
.........
}
```

```
using System;
interface Addition
{
  int Add();
}
interface Multiplication
{
int Mul();
}
class Compute:Addition,Multiplication
{
  int x,y;
  public Compute(int x, int y)
  {
    this.x=x;
    this.y=y;
   }
public int Add()                                    //implement Add()
{
  return(x+y);
}
public int Mul()                                     //implement Mul()
{
return(x*y);
}}
```

```
Class interfaceTest
{
  public static void Main()
   {
       Compute com=new Compute(10,20);              //Casting
       Addition add=(Addition)com;
       Console.WriteLine("sum is:"+add.Add());
      Multiplication mul=(Multiplication)com;       //Casting
      Console.WriteLine("product  is:"+mul.Mul());
    }
}
```

OUTPUT

sum is: 30
product is: 200

MULTIPLE IMPEMENTATION OF AN INTERFACE

Just as a class can implement multiple interfaces, one single interface can be implemented by multiple classes. We demonstrate this by the following example;

```
using System;
interface Area;
{
  double Compute(double x)
}
class Square:Area
{
  public double Compute(double x)
   {
     return(x*x);
   }
class Circle:Area
   {
     public double Compute(double x)
      {
        return(Math.PI*x*x);
      }
}
```

```
Class InterfaceTest
{
  public static void Main()
  {
    Square sqr=new Square();
    Circle cir=new Circle();
    Area area=(Area)sqr;
    Console.WriteLine("area of square is:"+area.Compute(10.0));
    area=(Area)cir;
    Console.WriteLine("area of circle is:"+area.Compute(10.0));
  }
}
```

OUTPUT

area of square is: 100
area of circle is: 314.159265

INTERFACES AND INHERITANCE:

We may encounter situations where the base class of a derived class implements an interface. In such situations, when an object of the derived class is converted to the interface type, the inheritance hierarchy is searched until it finds a class that has directly implemented the interface. We consider the following program:

```
using system;
interface Display
{
  void Print();
}
class B:Display            //implements Display
{
  public void Print()
   {
     Console.Writeline("base display");
   }
}
```

```
class D:B               //inherits B class
{
 public new void Print()
  {
    Console.Writeline("Derived display");
  }
}
class InterfaceTest3
{
 public static void Main()
 {
   D d=new D();
   d.Print();
   Display dis=(Display)d;
   dis.Print();
  }
}
```

OUTPUT:
Derived display
base display

The statement dis.Print() calls the method in the base class B but not in the derived class itself.

This is because the derived class doesn't implement the interface.

EXPLICIT INTERFACE IMPLEMENTATION

The main reason why c# doesn't support multiple inheritance is the problem of NAME COLLISION.

But the problem still persists in C# when we are implementing multiple interfaces.

For example:
let us consider the following scenario, there are two interfaces and an implementing class as follows:

```
interface i1
{
  void show();
}
interface i2
{
  void show();
}
class classone: i1,i2
{
  public void display()
  {
    ...
    ...
  }
```

The problem is that whether the class classone implements i1.show() or i2.show()??

This is ambiguous and the compiler will report an error.

To overcome this problem we have something called **EXPLICIT INTERFCAE IMPLEMENTATION**. This allows us to specify the name of the interface we want to implement.

We take the following program as an example:

```
using system;
intereface i1
{
  void show();
}
interface i2
{
  void show();
}
```

```
Class classone: i1,i2
{
  void i1.show()
   {
     Console.Writeline(“ implementing i1”);
   }
void i2.show()
 {
  Console.Writeline(“ implementing i2”);
 }
}
class interfaceImplement
 {
   public static void Main()
    {
      classone c1=new classone();
      i1 x=(i1)c1;
      x.show();
     i2 y=(i2)c1;
     y.show();
   }
}
```

OUTPUT: implementing i1
implementing i2


Merci
Gracias
Danke
Arigato