Multiple Inheritance

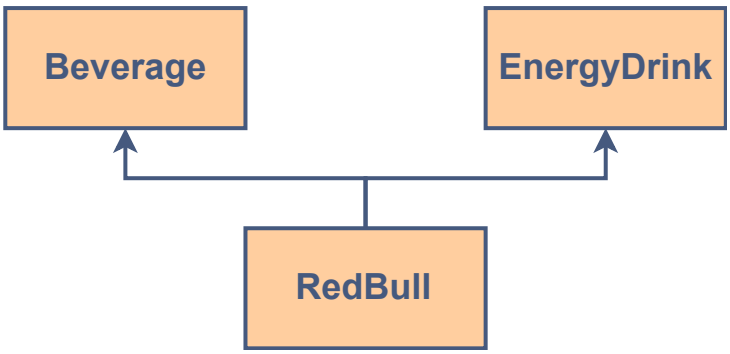
In this lesson, you'll learn what multiple inheritance is and how it can be implemented in C#.

WE'LL COVER THE FOLLOWING

- What Is Multiple Inheritance?
- How to Implement?
- An Example
- Interface vs Abstract Class

What Is Multiple Inheritance?

When a class is derived from more than a single base class, i.e., when a class has more than one immediate parent, it is an instance of **multiple Inheritance**. Let's take an example of Redbull:



RedBull is an energy drink as well as a beverage.

How to Implement?

As mentioned earlier, in C#, a class can't extend from more than one classes. So the question arises, "*How can we implement multiple inheritance?*"

The answer to the above question is an ***Interface***. In C#, *multiple inheritance* can be implemented using interfaces.

> A class can implement multiple interfaces and an interface can `extend` from multiple interfaces.

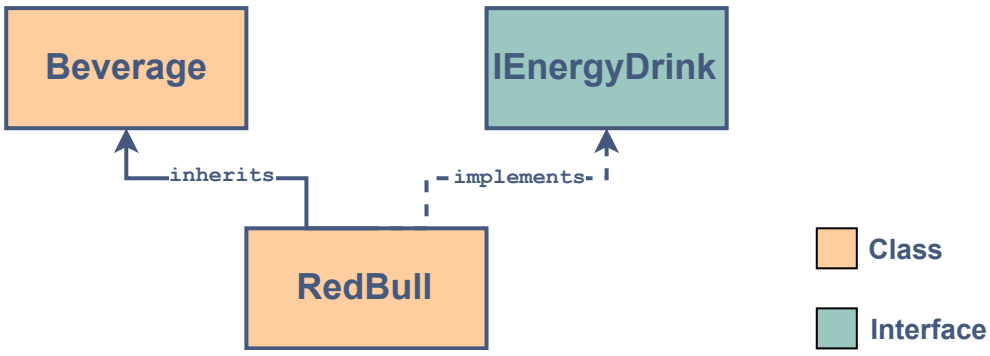
So, in this way, we can achieve multiple inheritance in C#. We already know that interfaces can only contain `public abstract` members with no implementation details. These members **must** be implemented by the class implementing this interface. This is not a pure form of inheritance, but nonetheless, it helps us in implementing functionalities from multiple sources.

An Example

Let's implement the example of `RedBull` given at the start of the lesson. This example can be implemented using:

- A base *class* named `Beverage`
- An *interface* named `IEnergyDrink`
- A `RedBull` class derived from `Beverage` and implementing `IEnergyDrink`

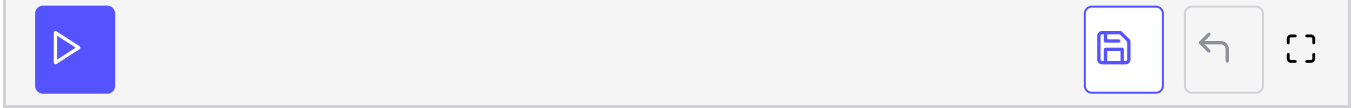
The above illustration then becomes:



Let's assume that the authorities have imposed an **additional $1 tax** on the sale of all kinds of energy drinks. Now we will have to take care of the addition of this tax to the selling price. Let's see how interfaces come in handy while implementing multiple inheritance. Below is the implementation:

```
abstract class Beverage // Base abstract class Beverage
{
    private string _name;
    private int _price;
```

```
    public Beverage(string name, int price)
    {
        this._name = name;
        this._price = price;
    }


    public int  Price{
      get {
        return this._price;
      }
    }
    public string Name {

      get {
        return this._name;
      }
    }
}

interface IEnergyDrink // Interface which imposes the tax method
{
    int CalculatePrice(int price);
}


class RedBull : Beverage, IEnergyDrink // Redbull class inheriting from both
{
    public RedBull(int price)
        : base("Redbull", price) {}

    // Any energy drink which implements IEnergyDrink will have to implement
    public int CalculatePrice(int price)
    {
        return (base.Price + 1);
    }

    public void PrintDetails()
    {
        Console.WriteLine("The name of the drink is: " + base.Name);
        Console.WriteLine("The price of the drink is: " + (CalculatePrice(base.Price)));

    }

}
class Program
{
    static void Main(string[] args)
    {
        RedBull redBull = new RedBull(5);
        redBull.PrintDetails();
    }
}
```



In the above example, we can see that the `RedBull` class is inheriting from

both the `IEnergyDrink` interface and `Beverage` class. Whenever an energy drink class will be implemented, it'll have inherited the properties of `Beverage` class and the functionality of `IEnergyDrink` interface.

Now that we've implemented multiple inheritance, let's take a look at the differences between an interface and an abstract class.

Interface vs Abstract Class

Interfaces and abstract classes are both used to achieve abstraction but with some key differences:

Interfaces	Abstract Classes
Can have abstract member(s) only	Can have concrete (non-abstract) & abstract member(s)
Support multiple inheritance	Don't support multiple inheritance
All members are `public abstract` by default	Can have members like non-abstract classes
Can't have constructors	Constructors can be defined

In the next lesson, there is a quiz to test your understanding of abstract classes and interfaces.