public class Sale

{

public double Price { get; set; }

public double Total { get; set; }

}

var Soda = new Sale();

var Potatoes = new Sale();

Soda.Price = 18.90;

Potatoes.Price = 12;

var NewSale = Soda.Price + Potatoes.Price;

Console.WriteLine(NewSale);

public class Sale

{

public Sale (double newPrice = 0)

{

this.UpdatePrice(newPrice);

}

public void UpdatePrice(double newPrice = 0)

{

if (newPrice != 0)

Price = newPrice;

}

public double Price { get; set; }

}

var Soda = new Sale(21.96);

Console.WriteLine(Soda.Price);

public class Product : Sale

{

public double NewPrice

{

set

{

UpdatePrice(value);

}

}

}

private double \_sale;

public static double Price;

public class Sale

{

public void UpdatePrice(double newPrice = 0)

{

if (newPrice != 0)

Price = newPrice;

}

public void UpdatePrice(double newPrice, double tax)

{

Price = newPrice + tax;

}

public double Price { get; set; }

}

using System;

using System.Collections.Generic;

// Base class

public class Animal

{

public virtual void Speak()

{

Console.WriteLine("The animal makes a sound.");

}

}

// Derived class 1

public class Dog : Animal

{

public override void Speak()

{

Console.WriteLine("The dog barks.");

}

}

// Derived class 2

public class Cat : Animal

{

public override void Speak()

{

Console.WriteLine("The cat meows.");

}

}

// Main program

public class Program

{

public static void Main()

{

// List of animals, but declared as the base class type

List<Animal> animals = new List<Animal>

{

new Dog(),

new Cat(),

new Animal()

};

// Each object behaves differently based on its actual class

foreach (var animal in animals)

{

animal.Speak();

}

}

}

interface IPerson

{

public string Name { get; set; }

public string FullName { get; set; }

}

class Person : IPerson

{

public string Name { get; set; }

public string FullName { get; set; }

}

For example, first we define an interface:

interface IPerson

{

public string Name { get; set; }

}

Then an implementation of the interface will be created:

class Person : IPerson

{

public void Name(string name)

{

Console.WriteLine($"What is your name? {name}");

}

}

Now we directly inject the dependency without the need to create an instance.

public class OrderName

{

private readonly IPerson \_person;

public OrderName(IPerson person)

{

\_person = person;

}

public void ProcessOrder(string name)

{

\_person.Name(name);

}

}

Finally, we construct the object by sending an instance of the Person object

IPerson person = new Person();

OrderName orderProcessor = new OrderName(person);

orderProcessor.ProcessOrder("My name is MAUI");