TutorialKart

C# Inheritance

C# Inheritance

Inheritance is an Object Oriented Programming concept in which a class can inherit the properties and methods of another class.

Usually, the inheriting class is called **derived class**. And the class which is inherited is called **base class**.

The derived class can use, extend or override the properties and methods of its base class.

Inheritance can be used when your use cases have categorical and sub-categorical objects.

Example 1 – C# Inheritance

This example demonstrates the usage of inheritance concept in C# programming.

Consider that our application deals with production line of cars. We will define a base class of `Car` and a derived class of `SportsCar`. A sports car can be considered as a sub-category of car in general.

We will have some properties and methods in `Car` class. And `SportsCar` class will override some of the methods, and have new methods of its own.

Program.cs

```
using System;

namespace CSharpExamples {
    class Car {
        public int chasis_number;
        public string name;
        public void addWheels(){
            Console.WriteLine("4 wheels added.");
        }
        public void addSeats(){
            Console.WriteLine("4 seats added.");
        }
    }

    class SportsCar: Car {
        public string name;
```

```
        public SportsCar(int chasis_number, string name){
            this.name = name;
            this.chasis_number = chasis_number;
        }
        public void addSeats(){
            Console.WriteLine("2 seats added.");
        }

        public void printDetails(){
            Console.WriteLine("\nCar Details\n------------");
            Console.WriteLine("Chasis Number : " + this.chasis_number);
            Console.WriteLine("Car Name : " + this.name);
        }
    }

    class Program {
        static void Main(string[] args) {
            SportsCar car1 = new SportsCar(254, "McLaren 570S");
            car1.addSeats();
            car1.addWheels();
            car1.printDetails();
        }
    }
}
```

Base Class: `Car` . `Car` has two properties: `chasis_number` and `name` . Car has two methods: `addWheels()` , `addSeats()` .

Derived Class: `SportsCar` . `SportsCar` inherits `Car` class. But `SportsCar` overrides the property `name` and method `addSeats()` . It has an additional method `printDetails()` . `SportsCar` class has constructor.

Output

```
2 seats added.
4 wheels added.

Car Details
------------
Chasis Number : 254
Car Name : McLaren 570S
```

Example 2 – About Overridden Properties

If you override a property, you cannot access that property in the method of base class.

Consider the above example. We have `printDetails()` method in derived class. And in `printDetails()` class, we are accessing the overridden property `name` . If we have the same `printDetails()` function in base class, we cannot access `name` property.

This is because, we can access properties of base class in derived class, but cannot access properties of

derived class in base class. As we have overridden the property `name`, we cannot access this in our base class.

Program.cs

```
using System;

namespace CSharpExamples {
    class Car {
        public int chasis_number;
        public string name;
        public void addWheels(){
            Console.WriteLine("4 wheels added.");
        }
        public void addSeats(){
            Console.WriteLine("4 seats added.");
        }
        public void printDetails(){
            Console.WriteLine("\nCar Details\n------------");
            Console.WriteLine("Chasis Number : " + this.chasis_number);
            Console.WriteLine("Car Name : " + this.name);
        }
    }

    class SportsCar: Car {
        public string name;

        public SportsCar(int chasis_number, string name){
            this.name = name;
            this.chasis_number = chasis_number;
        }
        public void addSeats(){
            Console.WriteLine("2 seats added.");
        }
    }

    class Program {
        static void Main(string[] args) {
            SportsCar car1 = new SportsCar(254, "McLaren 570S");
            car1.addSeats();
            car1.addWheels();
            car1.printDetails();
        }
    }
}
```

Output

```
2 seats added.
4 wheels added.

Car Details
------------
Chasis Number : 254
Car Name :
```

You might observe the following warnings before the actual output shown above.

```
Program.cs(21,23): warning CS0108: 'SportsCar.name' hides inherited member 'Car.name'. U
Program.cs(27,21): warning CS0108: 'SportsCar.addSeats()' hides inherited member 'Car.ad
Program.cs(6,23): warning CS0649: Field 'Car.name' is never assigned to, and will always
```

Conclusion

In this C# Tutorial, we have learned what Inheritance is in Object Oriented Programming, how to implement Inheritance in C# programming, and some of the implementations with example programs.

C# Tutorial

- C# Tutorial
- C# Basic Example
- C# Comments
- C# Variables
- C# Constants
- C# if, if-else
- C# switch
- C# while loop
- C# for loop
- C# foreach
- C# break
- C# continue
- C# struct
- C# enum
- C# String
- C# Array
- C# Command Line Arguments

C# Console Operations

- C# Write to Console
- C# Read from Console

C# Object Oriented Programming Concepts

- C# Class & Constructors
- C# Encapsulation
- C# Polymorphism
- C# Method Overloading
- **C# Interfaces**

C# String Operations

- C# String Length
- C# Substring

C# File Operations

- C# Read Text File
- C# Write to File
- C# Delete File
- C# Copy File

C# Exception Handling

- C# try-catch
- C# finally
- C# throw
- C# Custom Exception
- C# SystemException
- C# DivideByZeroException
- C# NullReferenceException
- C# InvalidCastException
- C# IOException
- C# FieldAccessException

C# Collections

- C# List
- C# SortedList
- C# HashSet

- C# SortedSet
- C# Stack
- C# Queue
- C# LinkedList
- C# Dicionary
- C# SortedDictionary

C# Errors [Solved]

- C# Error: Class does not contain a constructor that takes arguments