(T2)討論C#的ObjectOriented(物件導向)、Interface、BaseClass、SubClass  
CourseGUID: 29f1196a-1950-41a4-b9c1-dd13a9e92d92  
=======================================================================  
(T2)討論C#的ObjectOriented(物件導向)、Interface、BaseClass、SubClass  
=======================================================================  
1. Create New Project

-----------

2. Object Oriented

2.1. Interface : IVehicle.cs

2.3. Car.cs, Car : IVehicle

2.4. Toyota.cs, Toyota : Car

2.5. Mazda.cs, Mazda : Car

2.6. Flight.cs, Flight : IAircraft, IVehicle

2.7. CarFleet.cs

2.8. Jet.cs, Jet : Flight

2.9. Glider.cs, Glider : Flight

-----------

2. Main  
=======================================================================

1. Create New Project

File --> New --> Project... -->

Visual C# -->  **Console App** **(.Net Framework)** -->

Name: **Sample**







=============================================

2. Object Oriented

Diagram

Description automatically generated

2.1. Interface : IVehicle.cs

Graphical user interface, application, Word

Description automatically generated

// Namespace is like a folder which contains some classes.

namespace Sample

{

    // 1.

    // Interface is like an product booklet which contains

    // the standard actions that this product must be able to do.

    // Interface only contains the method signature without its body.

    // E.g.

    // IAircraft must be able to take off and land.

    // 2.

    // Interface can not contain fields.

    // 3.

    // The prefix of interface is "I"

    // 4.

    // a class can exten only one class and implement many Interface.

    // E.g.

    // public class ClassA : ClassB, InterfaceA, InterfaceB

    /// <summary>

    /// IVehicle must be able to move and stop

    /// </summary>

    public interface IVehicle

    {

        //string \_interfaceName = "IVehicle";  // Interface can not contain fields.

        /// <summary>

        /// IVehicle is moving

        /// </summary>

        /// <returns></returns>

        string Moving();

        /// <summary>

        /// IVehicle has stopped.

        /// </summary>

        /// <returns></returns>

        string Stop();

    }

}

2.2. Interface : IAircraft.cs

Graphical user interface, application, Word

Description automatically generated

namespace Sample

{

    // 1.

    // Interface is like an product booklet which contains

    // the standard actions that this product must be able to do.

    // Interface only contains the method signature without its body.

    // E.g.

    // IAircraft must be able to take off and land.

    // 2.

    // Interface can not contain fields.

    // 3.

    // The prefix of interface is "I"

    // 4.

    // a class can exten only one class and implement many Interface.

    // E.g.

    // public class ClassA : ClassB, InterfaceA, InterfaceB

    /// <summary>

    /// IAircraft must be able to take off and land.

    /// </summary>

    public interface IAircraft

    {

        //string \_interfaceName = "IAircraft";  // Interface can not contain fields.

        /// <summary>

        /// IAircraft is taking off.

        /// </summary>

        /// <returns></returns>

        string TakingOff();

        /// <summary>

        /// IAircraft is landing.

        /// </summary>

        /// <returns></returns>

        string Landing();

    }

}

2.3. Car.cs, Car : IVehicle

Graphical user interface, application

Description automatically generated

// Namespace is like a folder which contains some classes.

namespace Sample

{

    // 1.

    // public / protected / private

    // Reference:

    // <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/accessibility-levels>

    // Accessibility Levels includes several levels.

    // Here, we only discuss, public, protected, and private.

    // public means access is not restricted.

    // protected means access is limited to the containing class or types derived from the containing class.

    // private means access is limited to the containing type.

    // 2.

    // class is like a blueprint or template.

    // object is a single instance of the class.

    public class Car : IVehicle

    {

        //--------------------------------------------------------------

        // This can be called as Class Member, Field, global variable.

        // Most people called this as "Field".

        // Field is like a Database Table Column to store the data of the object.

        private string \_make;

        private string \_type;

        private string \_registration;

        private string \_year;

        //--------------------------------------------------------------

        //The constructor is a special method.

        //Whenever a class or struct is created, its constructor is called

        /// <summary>

        /// The constructor of car.

        /// </summary>

        /// <param name="make"></param>

        /// <param name="type"></param>

        /// <param name="registration"></param>

        /// <param name="year"></param>

        /// <param name="currentValue"></param>

        public Car(string make, string type, string registration, string year, double currentValue)

        {

            \_make = make;

            \_type = type;

            \_registration = registration;   // set value dirrectly to the field.

            \_year = year;   // set value dirrectly to the field.

            CurrentValue = currentValue;    // set the field value by its property.

        }

        //--------------------------------------------------------------

        // Properties is special method to replace get and set.

        // Year Property can replace GetYear() and SetYear()

        public string Year

        {

            get { return \_year; }

            // this is the keyword, means current object

            set { this.\_year = value; }

        }

        // CurrentValue Property can replace GetValue() and SetValue()

        public double CurrentValue { get; set; }

        // Make Property can replace GetMake()

        public string Make { get { return \_make; } }

        // Type Property can replace GetType()

        public string Type { get { return \_type; } }

        //--------------------------------------------------------------

        // 1.

        //Method is a set of logic processes.

        //Method is like an action which this object can do.

        //E.g. Car can move and stop.

        //2.

        //Only virtual method can be overrided in the sub-class.

        /// <summary>

        /// Get the \_registration

        /// </summary>

        /// <returns>\_registration</returns>

        public string GetRegistration()

        {

            return \_registration;

        }

        /// <summary>

        /// Set the \_registration

        /// </summary>

        /// <param name="registration">registration string</param>

        public void SetRegistration(string registration)

        {

            \_registration = registration;

        }

        /// <summary>

        /// Return full car information

        /// </summary>

        /// <returns>full car information</returns>

        public override string ToString()

        {

            //return String.Format("Car Make: {0}\n" +

            //                     "Car Type: {1}\n" +

            //                     "Car Registration: {2}\n" +

            //                     "Car Year: {3}\n" +

            //                     "Current Value: {4}", \_make, \_type, \_registration, \_year, CurrentValue);

            return $"Car Make: {\_make}\n" +

                $"Car Type: {\_type}\n" +

                $"Car Registration: {\_registration}\n" +

                $"Car Year: {\_year}\n" +

                $"Current Value: {CurrentValue}";

        }

        /// <summary>

        /// IVehicle is moving

        /// </summary>

        /// <returns></returns>

        public virtual string Moving()

        {

            return "Car is moving.";

        }

        /// <summary>

        /// IVehicle has stopped.

        /// </summary>

        /// <returns></returns>

        public string Stop()

        {

            return "Car has stopped.";

        }

    }

}

2.4. Toyota.cs, Toyota : Car

Graphical user interface, application

Description automatically generated

// Namespace is like a folder which contains some classes.

namespace Sample

{

    // 1.

    // class is like a blueprint or template.

    // object is a single instance of the class.

    // 2.

    // Toyota : Car

    // means Toyota extend or implement Car.

    // We can say Car is the parent class of Toyota.

    // Toyota is a sub-Class of Car.

    // Toyota succeed all members, properties, methods

    // from its parent class, Car.

    public class Toyota : Car

    {

        //--------------------------------------------------------------

        //The constructor is a special method.

        //Whenever a class or struct is created, its constructor is called

        public Toyota(string type, string registration, string year, double currentValue) : base("Toyota", type, registration, year, currentValue)

        {

        }

        //--------------------------------------------------------------

        // 1.

        //Method is a set of logic processes.

        //Method is like an action which this object can do.

        //E.g. Car can move and stop.

        //2.

        //Only virtual method can be overrided in the sub-class.

        /// <summary>

        /// An action or method which ONLY Toyota can do.

        /// </summary>

        /// <returns></returns>

        public string OnlyToyotaCanDo()

        {

            return "This is an action or method which ONLY " + this.Make + " can do.";

        }

        /// <summary>

        /// IVehicle is moving

        /// </summary>

        /// <returns></returns>

        public override string Moving()

        {

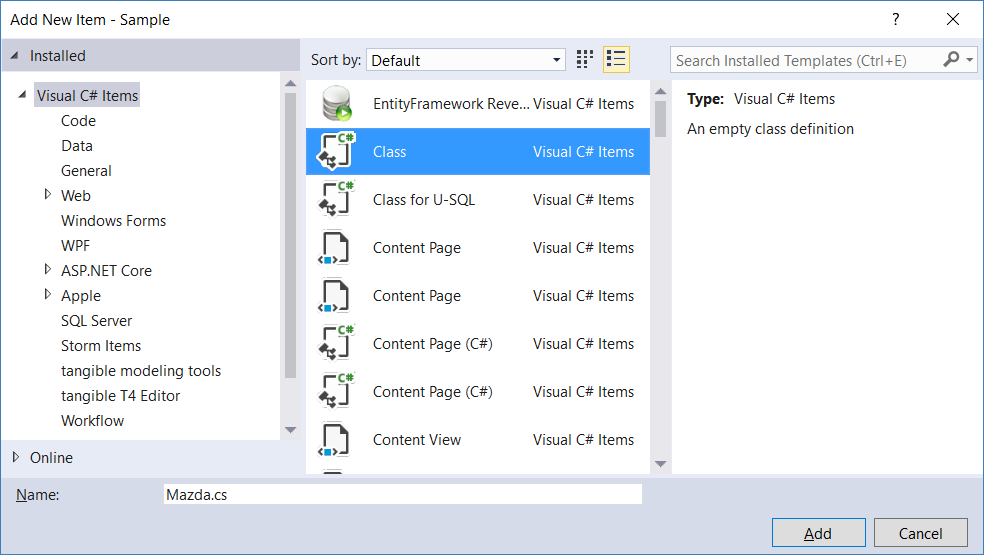
            return this.Make + " Car is moving.";

        }

    }

}

2.5. Mazda.cs, Mazda : Car



// Namespace is like a folder which contains some classes.

namespace Sample

{

    // 1.

    // class is like a blueprint or template.

    // object is a single instance of the class.

    // 2.

    // Mazda : Car

    // means Mazda extend or implement Car.

    // We can say Car is the parent class of Mazda.

    // Mazda is a sub-Class of Car.

    // Mazda succeed all members, properties, methods

    // from its parent class, Car.

    public class Mazda : Car

    {

        //--------------------------------------------------------------

        //The constructor is a special method.

        //Whenever a class or struct is created, its constructor is called

        public Mazda(string type, string registration, string year, double currentValue) : base("Mazda", type, registration, year, currentValue)

        {

        }

        //--------------------------------------------------------------

        // 1.

        //Method is a set of logic processes.

        //Method is like an action which this object can do.

        //E.g. Car can move and stop.

        //2.

        //Only virtual method can be overrided in the sub-class.

        /// <summary>

        /// An action or method which ONLY Mazda can do.

        /// </summary>

        /// <returns></returns>

        public string OnlyMazdaCanDo()

        {

            return "This is an action or method which ONLY " + this.Make + " can do.";

        }

    }

}

2.6. Flight.cs, Flight : IAircraft, IVehicle

Graphical user interface, application

Description automatically generated

// Namespace is like a folder which contains some classes.

namespace Sample

{

    // a class can exten only one class and implement many Interface.

    // E.g.

    // public class ClassA : ClassB, InterfaceA, InterfaceB

    public class Flight : IAircraft, IVehicle

    {

        //--------------------------------------------------------------

        // This can be called as Class Member, Field, global variable.

        // Most people called this as "Field".

        // Field is like a Database Table Column to store the data of the object.

        private string \_type;

        private string \_year;

        //--------------------------------------------------------------

        //The constructor is a special method.

        //Whenever a class or struct is created, its constructor is called

        /// <summary>

        /// The constructor of Flight.

        /// </summary>

        /// <param name="type"></param>

        /// <param name="year"></param>

        /// <param name="currentValue"></param>

        public Flight(string type, string year, double currentValue)

        {

            \_type = type;

            \_year = year;   // set value dirrectly to the field

            CurrentValue = currentValue;    // set the field value by its property.

        }

        //--------------------------------------------------------------

        // Properties is special method to replace get and set.

        // Year Property can replace GetYear() and SetYear()

        public string Year

        {

            get { return \_year; }

            // this is the keyword, means current object

            set { this.\_year = value; }

        }

        // CurrentValue Property can replace GetValue() and SetValue()

        public double CurrentValue { get; set; }

        // Make Property can replace GetMake()

        public string Type { get { return \_type; } }

        //--------------------------------------------------------------

        //1.

        //Method is a set of logic processes.

        //Method is like an action which this object can do.

        //E.g. Flight can take off and land.

        //2.

        //Only virtual method can be overrided in the sub-class.

        /// <summary>

        /// IAircraft is taking off.

        /// </summary>

        /// <returns></returns>

        public virtual string TakingOff()

        {

            return "Flight is taking off.";

        }

        /// <summary>

        /// IAircraft is landing.

        /// </summary>

        /// <returns></returns>

        public virtual string Landing()

        {

            return "Flight is landing.";

        }

        /// <summary>

        /// IVehicle is moving

        /// </summary>

        /// <returns></returns>

        public string Moving()

        {

            return "Flight is moving.";

        }

        /// <summary>

        /// IVehicle has stopped.

        /// </summary>

        /// <returns></returns>

        public string Stop()

        {

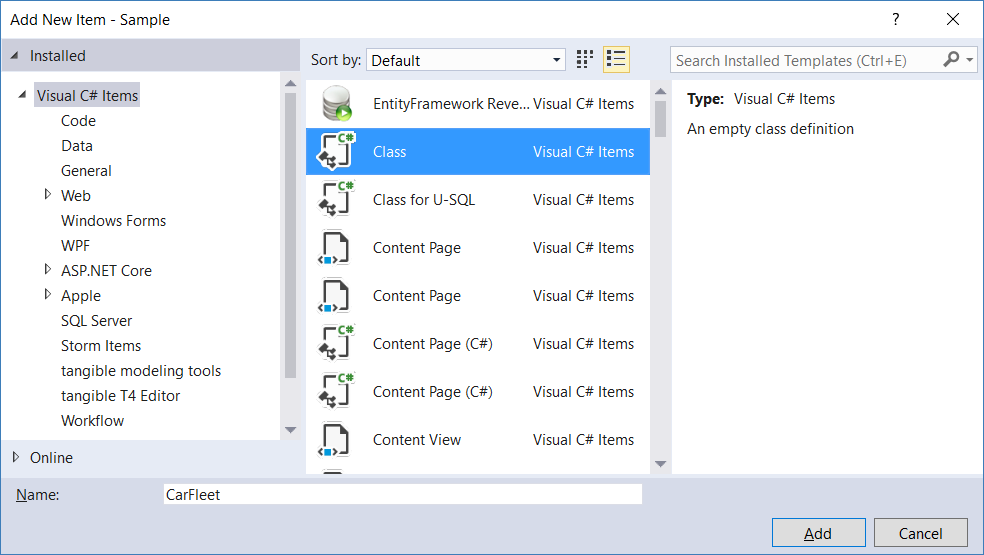
            return "Flight has stopped.";

        }

    }

}

2.7. CarFleet.cs



namespace Sample

{

    public class CarFleet

    {

        //--------------------------------------------------------------

        // This can be called as Class Member, Field, global variable.

        // Most people called this as "Field".

        // Field is like a Database Table Column to store the data of the object.

        // an array of \_cars which can contain 100 car objects.

        //Max number of car quanty is 100.

        private Car[] \_cars;

        // The current index of car array represents the current car quantity in array.

        // In the biginning, it should be 0.

        private int \_qtyOfCarInArray;

        //--------------------------------------------------------------

        //The constructor is a special method.

        //Whenever a class or struct is created, its constructor is called

        /// <summary>

        /// The constructor.

        /// </summary>

        public CarFleet()

        {

            // an array of \_cars which can contain 100 car objects.

            //Max number of car quanty is 100.

            \_cars = new Car[100];

            // The current index of car array represents the current car quantity in array.

            // In the biginning, it should be 0.

            \_qtyOfCarInArray = 0;

        }

        //--------------------------------------------------------------

        // 1.

        //Method is a set of logic processes.

        //Method is like an action which this object can do.

        //E.g. Car can move and stop.

        //2.

        //Only virtual method can be overrided in the sub-class.

        /// <summary>

        /// Add the car into the next free slot in the car array.

        /// </summary>

        /// <param name="car">The car to add</param>

        public void Add(Car car)

        {

            // \_qtyOfCarInArray field tracks where

            // the next free slot in the array is and

            // increments it after a car has been added.

            \_cars[\_qtyOfCarInArray] = car;

            \_qtyOfCarInArray++;

        }

        /// <summary>

        /// Summing up the value of each car in the car array.

        /// </summary>

        /// <returns>Return the Sum value of each car in the car array.</returns>

        public double SumFleetValue()

        {

            double total = 0;

            for (int i = 0; i < \_qtyOfCarInArray; i++)

            {

                total += \_cars[i].CurrentValue; // total = total + \_cars[i].CurrentValue;

            }

            return total;

        }

        /// <summary>

        /// Output parameter for the car Min Value and

        /// car Max Value from the car array.

        /// </summary>

        /// <param name="leastValue">Car least Value from the car array</param>

        /// <param name="highestValue">Car highest Value from the car array</param>

        public void Statistics(out double leastValue, out double highestValue)

        {

            leastValue = \_cars[0].CurrentValue;

            highestValue = \_cars[0].CurrentValue;

            for (int i = 0; i < \_qtyOfCarInArray; i++)

            {

                if (\_cars[i].CurrentValue < leastValue)

                    leastValue = \_cars[i].CurrentValue;

                else if (\_cars[i].CurrentValue > highestValue)

                    highestValue = \_cars[i].CurrentValue;

            }

        }

        /// <summary>

        /// Get the cars by its year.

        /// </summary>

        /// <param name="year">The specific car year.</param>

        /// <returns>The cars by its year.</returns>

        public Car[] GetCars(string year)

        {

            //Count how many cars in the car array are for the specified year.

            int count = 0;

            for (int i = 0; i < \_qtyOfCarInArray; i++)

            {

                if (\_cars[i].Year == year)

                    count++;

            }

            // Create a new array, carsYears, with this size, count.

            Car[] carsYears = new Car[count];

            // copy in the cars for the specified year into new arrray, carsYears.

            int index = 0;

            for (int i = 0; i < \_qtyOfCarInArray; i++)

            {

                if (\_cars[i].Year == year)

                {

                    carsYears[index] = \_cars[i];

                    index++;

                }

            }

            // return the array.

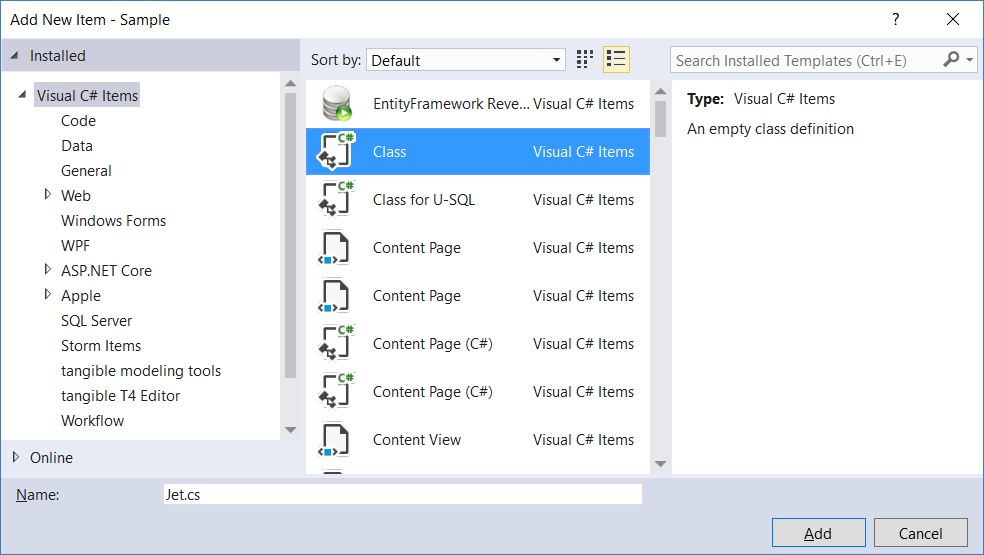
            return carsYears;

        }

    }

}

2.8. Jet.cs, Jet : Flight



// Namespace is like a folder which contains some classes.

namespace Sample

{

    public class Jet : Flight

    {

        //--------------------------------------------------------------

        //The constructor is a special method.

        //Whenever a class or struct is created, its constructor is called

        public Jet(string year, double currentValue) : base("Jet", year, currentValue)

        {

        }

        //--------------------------------------------------------------

        // 1.

        //Method is a set of logic processes.

        //Method is like an action which this object can do.

        //E.g. Car can move and stop.

        //2.

        //Only virtual method can be overrided in the sub-class.

        /// <summary>

        /// An action or method which ONLY Jet can do.

        /// </summary>

        /// <returns></returns>

        public string OnlyJetCanDo()

        {

            return "This is an action or method which ONLY " + this.Type + " can do.";

        }

        /// <summary>

        /// IAircraft is taking off.

        /// </summary>

        /// <returns></returns>

        public override string TakingOff()

        {

            return "Flight is taking off.";

        }

        /// <summary>

        /// IAircraft is landing.

        /// </summary>

        /// <returns></returns>

        public override string Landing()

        {

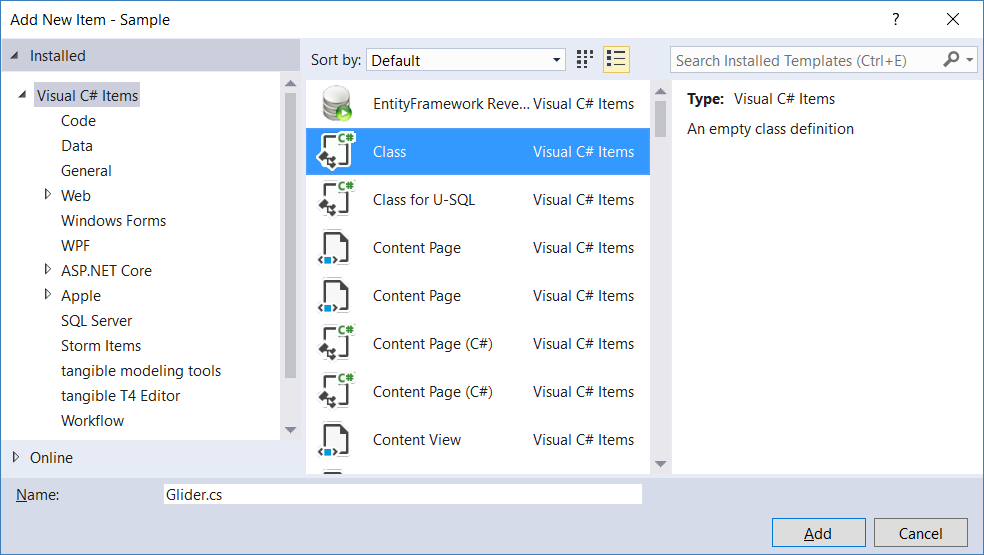
            return "Flight is landing.";

        }

    }

}

2.9. Glider.cs, Glider : Flight



// Namespace is like a folder which contains some classes.

namespace Sample

{

    public class Glider : Flight

    {

        //--------------------------------------------------------------

        //The constructor is a special method.

        //Whenever a class or struct is created, its constructor is called

        public Glider(string year, double currentValue) : base("Glider", year, currentValue)

        {

        }

        //--------------------------------------------------------------

        // 1.

        //Method is a set of logic processes.

        //Method is like an action which this object can do.

        //E.g. Car can move and stop.

        //2.

        //Only virtual method can be overrided in the sub-class.

        /// <summary>

        /// An action or method which ONLY Glider can do.

        /// </summary>

        /// <returns></returns>

        public string OnlyGliderCanDo()

        {

            return "This is an action or method which ONLY " + this.Type + " can do.";

        }

    }

}

=============================================

2. Main

using System;

// Namespace is like a folder which contains some classes.

namespace Sample

{

    class Program

    {

        static void Main(string[] args)

        {

            //1.

            //Create a detail for Honda CRV and print its detail.

            Console.WriteLine("CarA ----------------------------------");

            Car carA = new Car("Honda", "Crv", "RegistrationA", "2014", 21000);

            Console.WriteLine(carA);

            Console.WriteLine(carA.Moving());

            Console.WriteLine(carA.Stop());

            //2.

            //Create a detail for Toyota Corolla and print its detail.

            Console.WriteLine("CarB ----------------------------------");

            // Class ObjectName = new Class

            // E.g.

            // Use Toyota class as the blueprint and create an instance object of Toyota.

            // The instance object name is carB.

            Toyota carB = new Toyota("Prius", "RegistrationB", "2014", 23000);

            Console.WriteLine(carB);

            Console.WriteLine(carB.Moving());

            Console.WriteLine(carB.Stop());

            Console.WriteLine(carB.OnlyToyotaCanDo());

            //3.

            //Create a detail for Toyota Corolla and print its detail.

            Console.WriteLine("CarC ----------------------------------");

            // We can also use Toyota's parents class, Car, to create Toyota instance object.

            Car carC = new Toyota("Corolla", "RegistrationC", "2017", 25000);

            Console.WriteLine(carC);

            Console.WriteLine(carC.Moving());

            Console.WriteLine(carC.Stop());

            //// Error! Because we already cast Toyota to Car type variable, carC.

            //// Thus, carC has no "OnlyToyotaCanDo()" method.

            //Console.WriteLine(carC.OnlyToyotaCanDo());

            Console.WriteLine(((Toyota)carC).OnlyToyotaCanDo());

            //4.

            Console.WriteLine("CarD ----------------------------------");

            // We can also use Toyota's parents Interface, IVehicle, to create Toyota instance object.

            IVehicle carD = new Toyota("Camry", "RegistrationD", "2017", 28000);

            Console.WriteLine(carD);

            Console.WriteLine(carD.Moving());

            Console.WriteLine(carD.Stop());

            //// Error! Because we already cast Toyota to Car type variable, carD.

            //// Thus, carD has no "OnlyToyotaCanDo()" method.

            //Console.WriteLine(carD.OnlyToyotaCanDo());

            Console.WriteLine(((Toyota)carD).OnlyToyotaCanDo());

            //5.

            Console.WriteLine("CarE ----------------------------------");

            // We can also use "var" to create Toyota instance object.

            var carE = new Toyota("Prius C", "RegistrationE", "2016", 20000);

            Console.WriteLine(carE);

            Console.WriteLine(carE.Moving());

            Console.WriteLine(carE.Stop());

            Console.WriteLine(carE.OnlyToyotaCanDo());

            //6.

            Console.WriteLine("CarF ----------------------------------");

            // We can also use "var" to create Toyota instance object.

            var carF = new Mazda("Three", "RegistrationF", "2016", 25000);

            Console.WriteLine(carF);

            Console.WriteLine(carF.Moving());

            Console.WriteLine(carF.Stop());

            Console.WriteLine(carF.OnlyMazdaCanDo());

            //7.

            Console.WriteLine("CarG ----------------------------------");

            // We can also use "var" to create Toyota instance object.

            Mazda carG = new Mazda("Six", "RegistrationG", "2016", 29000);

            Console.WriteLine(carG);

            Console.WriteLine(carG.Moving());

            Console.WriteLine(carG.Stop());

            Console.WriteLine(carG.OnlyMazdaCanDo());

            //8.

            Console.WriteLine("CarFleet ==================================");

            // Add cars to CarFleet object.

            CarFleet carFleet = new CarFleet();

            carFleet.Add(carA);

            carFleet.Add(carB);

            carFleet.Add(carC);

            carFleet.Add((Car)carD);

            carFleet.Add(carE);

            carFleet.Add(carF);

            carFleet.Add(carG);

            // Print the sum value.

            Console.WriteLine("Total sum value : {0}", carFleet.SumFleetValue());

            // print the max and min value.

            double max, min;

            carFleet.Statistics(out min, out max);

            Console.WriteLine("The most expensive car value : {0}", max);

            Console.WriteLine("The cheapest car value : {0}", min);

            // print the cars by its year.

            Car[] carYears = carFleet.GetCars("2016");

            foreach (Car car in carYears)

            {

                Console.WriteLine("CarFleet {0}, {1} ----------------------", car.Make, car.Type);

                Console.WriteLine(car);

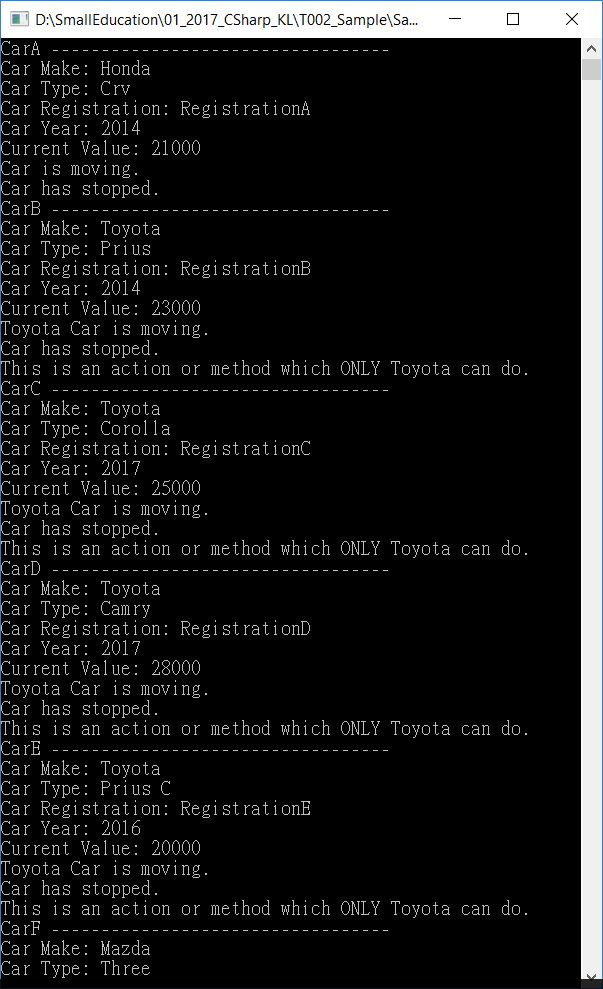
            }

            Console.ReadLine();

        }

    }

}



Text

Description automatically generated