C# Object Oriented Programming

Four pillars of Object Oriented Programming

- **Abstraction**
  - *What – not how*
- **Encapsulation**
  - *Avoid direct access to data (data hiding)*
- **Inheritance**
  - *Reusing and adding new abilities*
- **Polymorphism**
  - *Objects with different types can be accessed through the same interface*
  - *Same name of function, difference in behaviors*

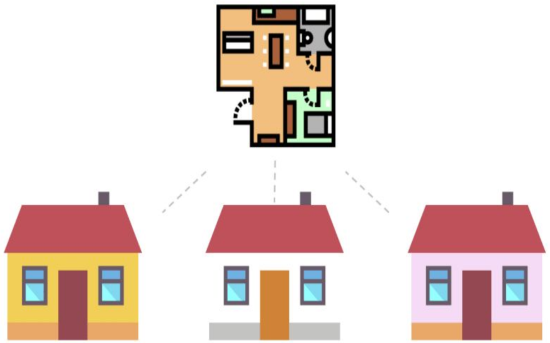
Core concepts

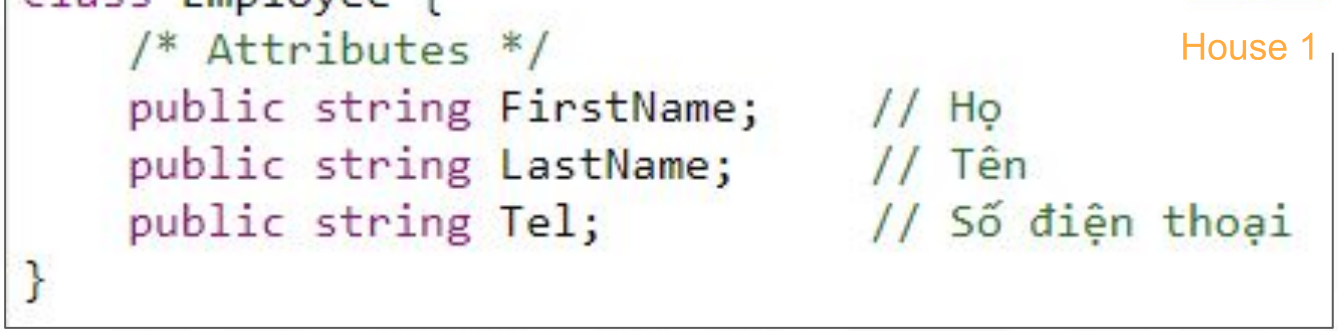
- ❑ Class & Object, UML Class diagram
- ❑ Override & Overloading
- ❑ Abstract class
- ❑ Interface

Class

A blueprint to create objects

Class House

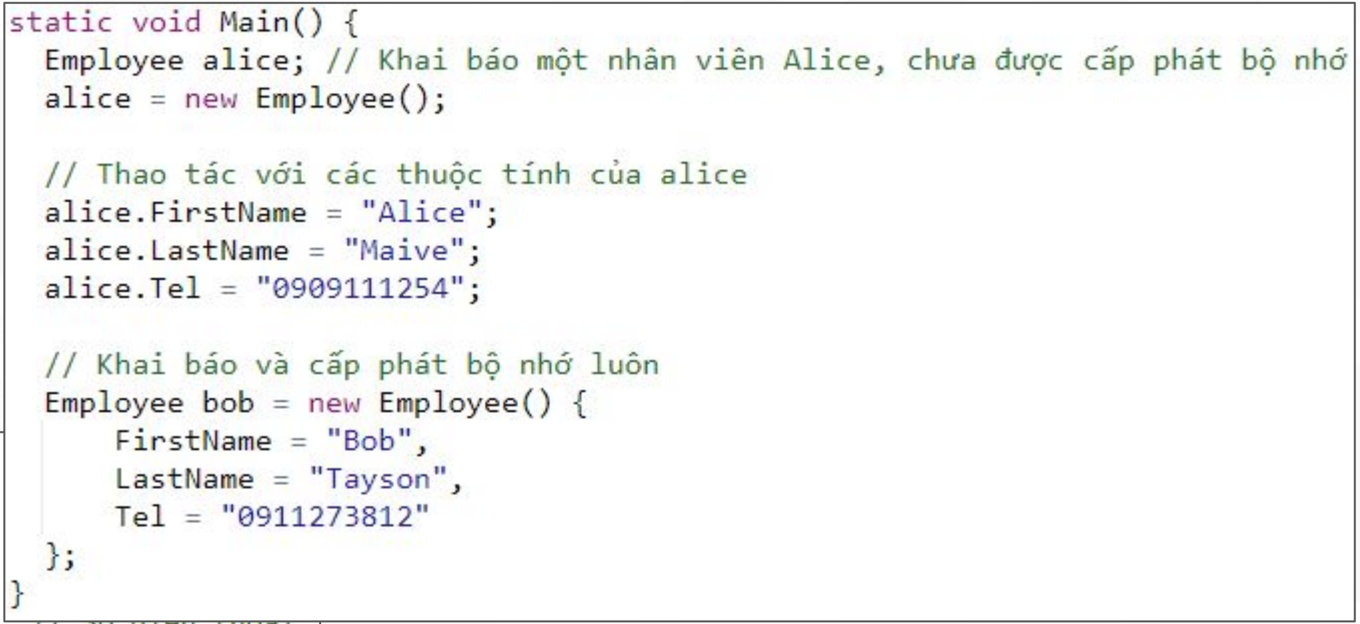




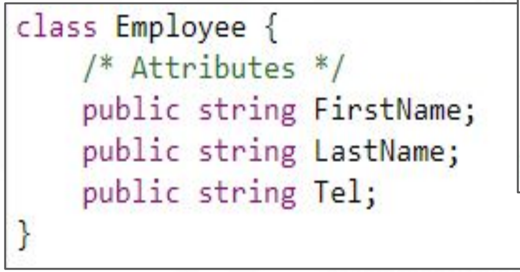
```
class Employee {
    /* Attributes */
    public string FirstName;     // Họ
    public string LastName;      // Tên
    public string Tel;           // Số điện thoại
}
```

Object

A specific instance created from a class



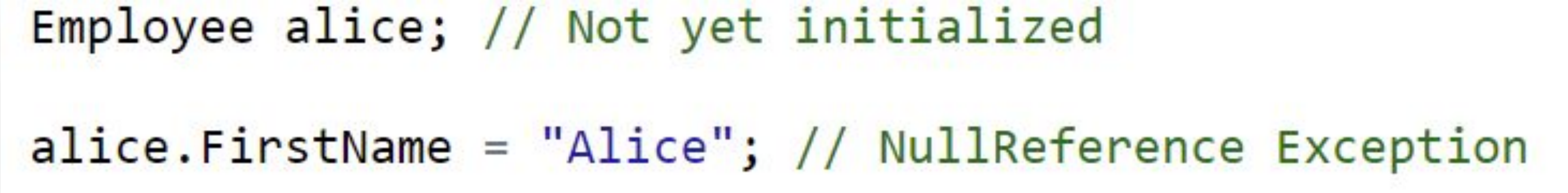
```
static void Main() {
  Employee alice; // Khai báo một nhân viên Alice, chưa được cấp phát bộ nhớ
  alice = new Employee();

  // Thao tác với các thuộc tính của alice
  alice.FirstName = "Alice";
  alice.LastName = "Maive";
  alice.Tel = "0909111254";

  // Khai báo và cấp phát bộ nhớ luôn
  Employee bob = new Employee() {
      FirstName = "Bob",
      LastName = "Tayson",
      Tel = "0911273812"
  };
}
```



```
class Employee {
    /* Attributes */
    public string FirstName;
    public string LastName;
    public string Tel;          // Số điện thoại
}
```

How many classes are there? How many instances are there?

Common misunderstanding & error for beginners



```
Employee alice; // Not yet initialized

alice.FirstName = "Alice"; // NullReference Exception
```

Destructor

No need because we have an automatic **Garbage Collector**

What does a class have?

- ❏ Attributes (data)
- ❏ Behaviors (functions)

Attributes: weight, height

Behaviors: eat, sleep, run

When a cat **eats**, what changes?
When a cat **sleep**, what changes?

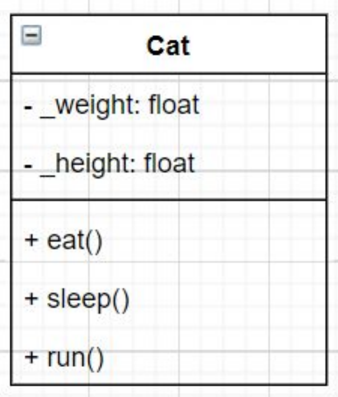
Behaviors can change attributes

eat

sleep

Represent a cat with UML

❑ Universal Modeling Language



Cat

- _weight: float

- _height: float

+ eat()

+ sleep()

+ run()

Attributes

Behaviors

Note

1. Class is a blueprint of an object
2. Object is an instance of a class
3. A class has attributes & behaviors
4. Behaviors can change an object’s attributes
5. Use UML to model a class
6. Abstraction: focus on what is important

In-class exercise

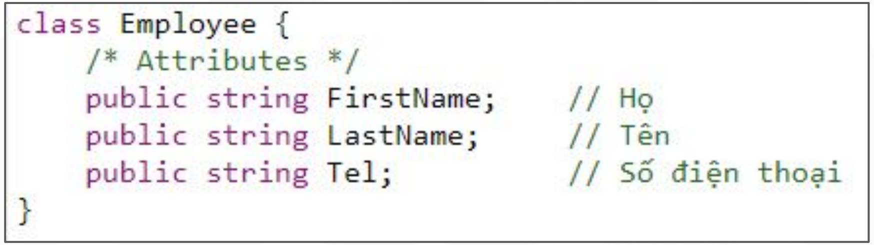
Create UML diagram using draw.io (or your favourite) for

1. A soldier that has Hit points and Mana, can attack and defend
2. An Autobot transformer that can toggle between vehicle form and robot form, fly, fight and escape. Its life depends on its energy in W (wattage)

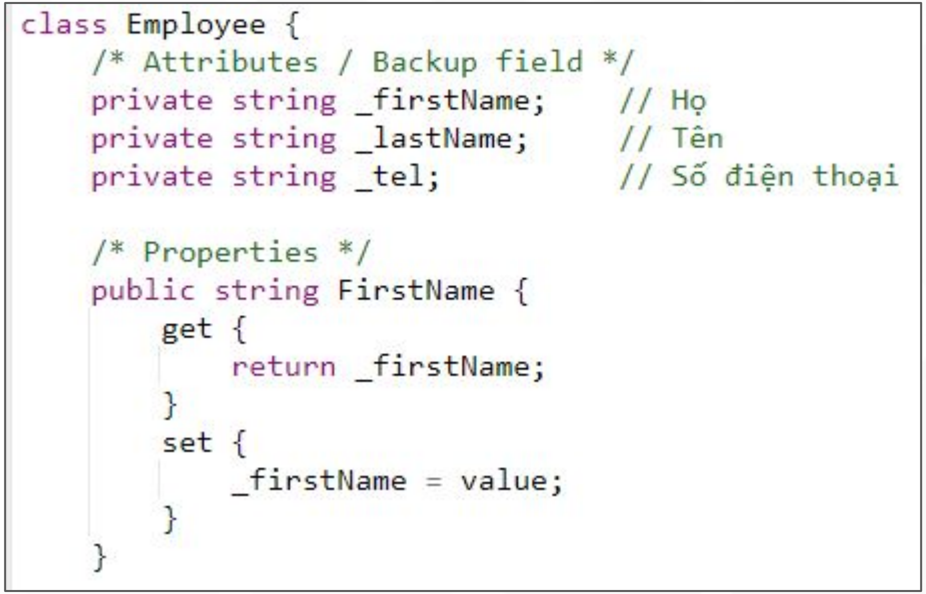
Submit back the image in this format: StudentID.png/jpg

Encapsulation - Getter & Setter

Avoid direct access to attribute

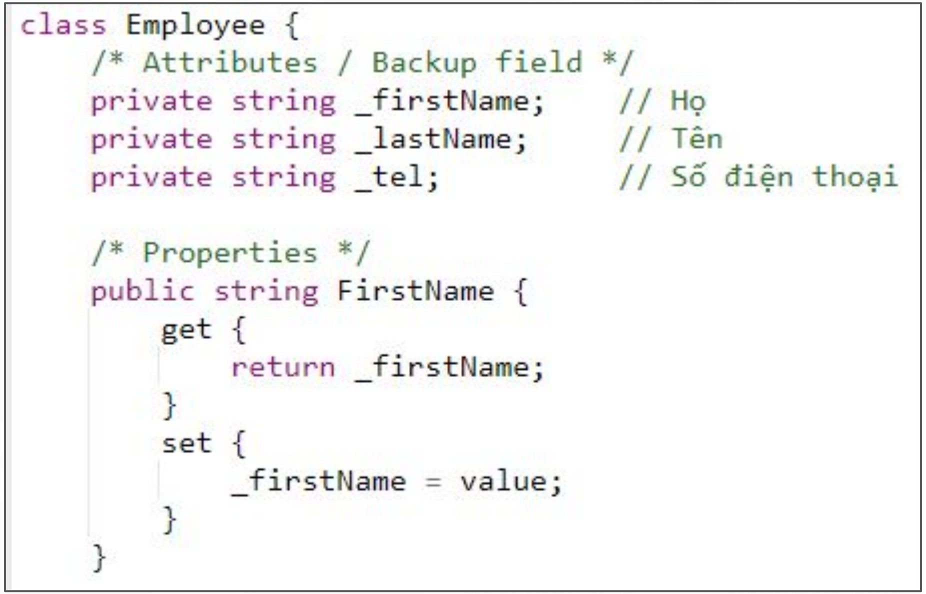


```
class Employee {
    /* Attributes */
    public string FirstName;    // Họ
    public string LastName;     // Tên
    public string Tel;          // Số điện thoại
}
```

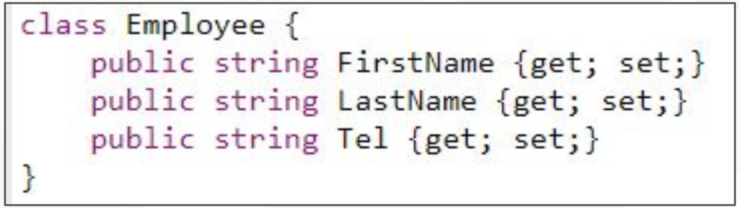


```
class Employee {
    /* Attributes / Backup field */
    private string _firstName;    // Họ
    private string _lastName;     // Tên
    private string _tel;          // Số điện thoại

    /* Properties */
    public string FirstName {
        get {
            return _firstName;
        }
        set {
            _firstName = value;
        }
    }
```

Shortcut for encapsulation



```
class Employee {
    /* Attributes / Backup field */
    private string _firstName;     // Họ
    private string _lastName;      // Tên
    private string _tel;           // Số điện thoại

    /* Properties */
    public string FirstName {
        get {
            return _firstName;
        }
        set {
            _firstName = value;
        }
    }
```



```
class Employee {
    public string FirstName {get; set;}
    public string LastName {get; set;}
    public string Tel {get; set;}
}
```

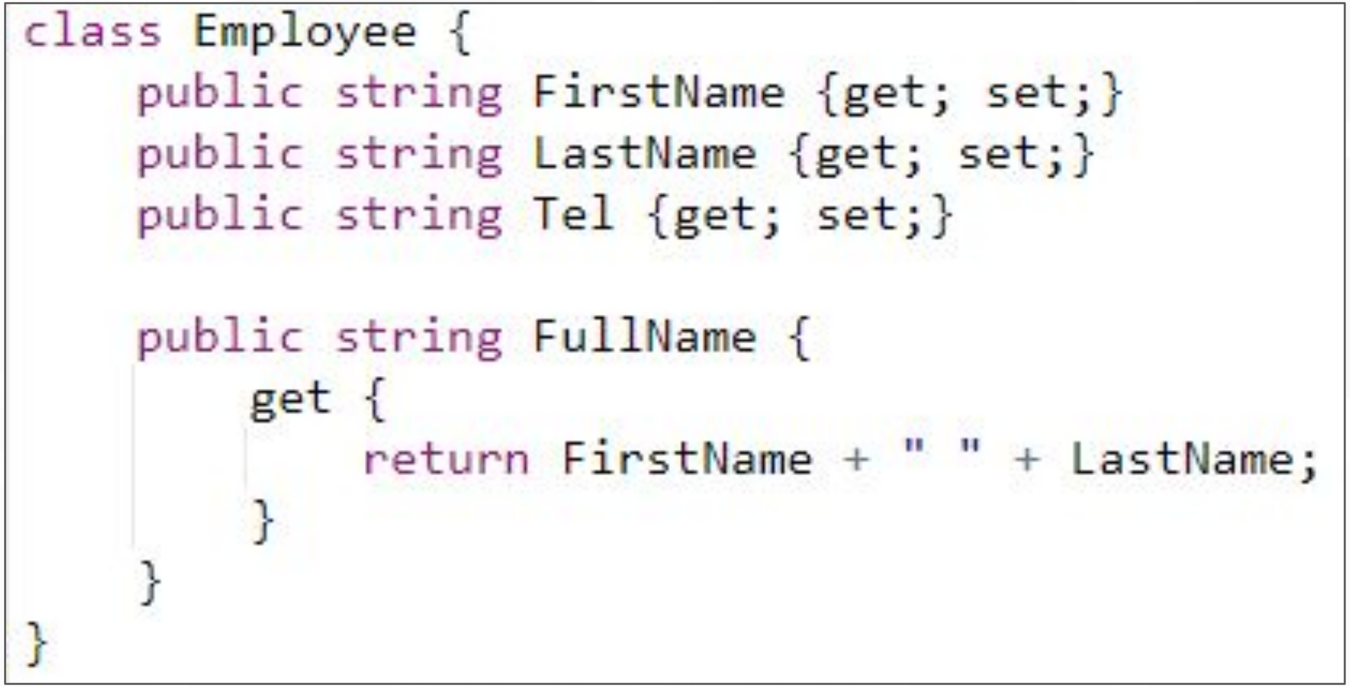
Why should we use getter & setter?

1. **May be adding business logic later**
   - ❏ For example: your account has a balance of 200.000, you cannot withdraw 300.000
   - ❏ Isolation, good for maintenance

   Derivative properties
2. **UI programming: databinding!**

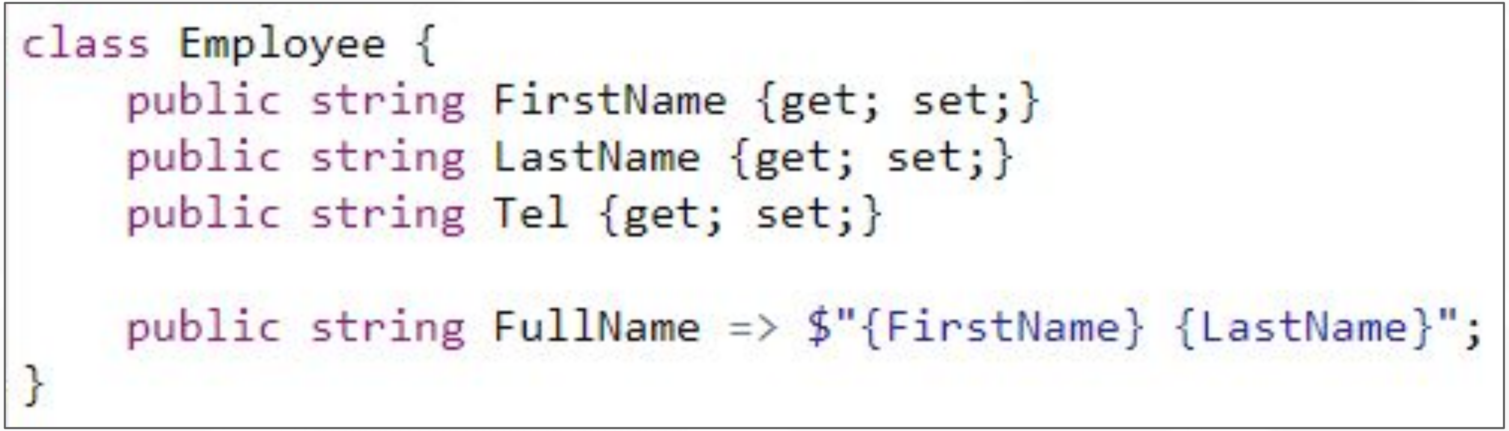
   UI elements automatically update based on the databound variable

Derivative property



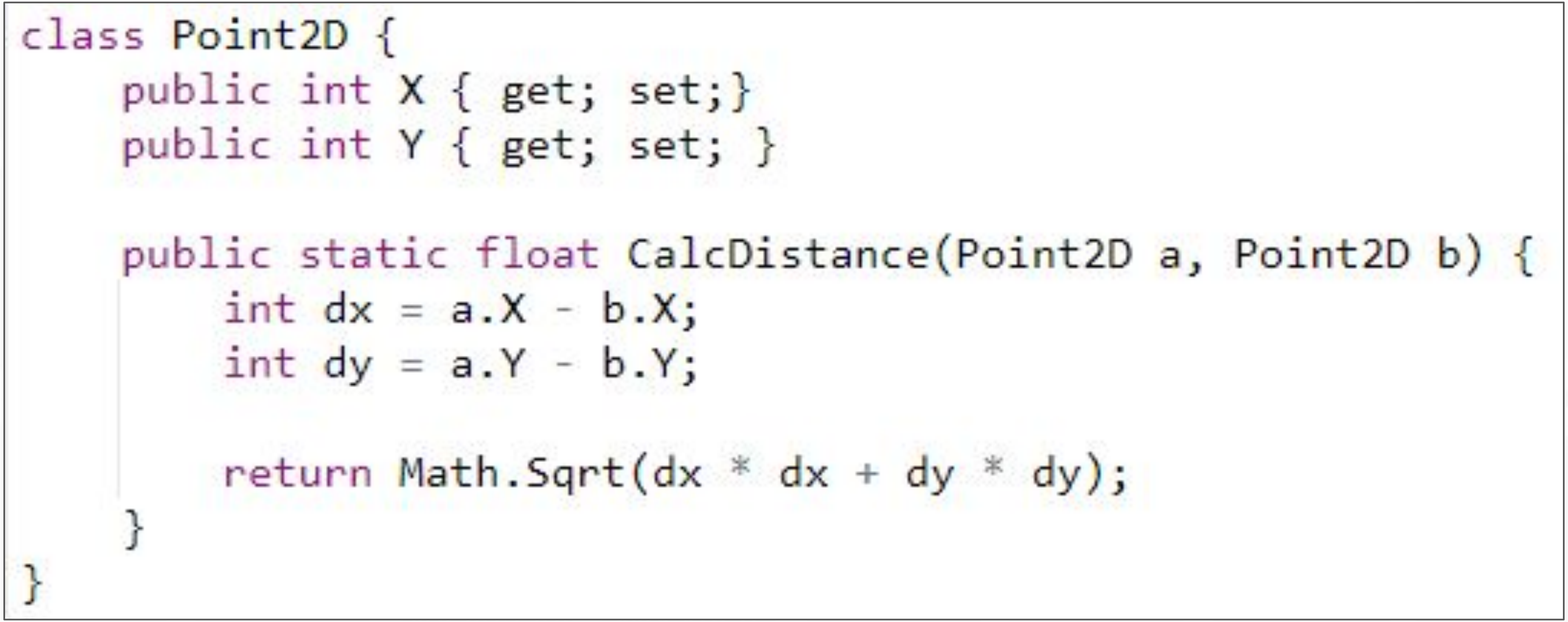
```
class Employee {
    public string FirstName {get; set;}
    public string LastName {get; set;}
    public string Tel {get; set;}

    public string FullName {
        get {
            return FirstName + " " + LastName;
        }
    }
}
```

Lambda expression



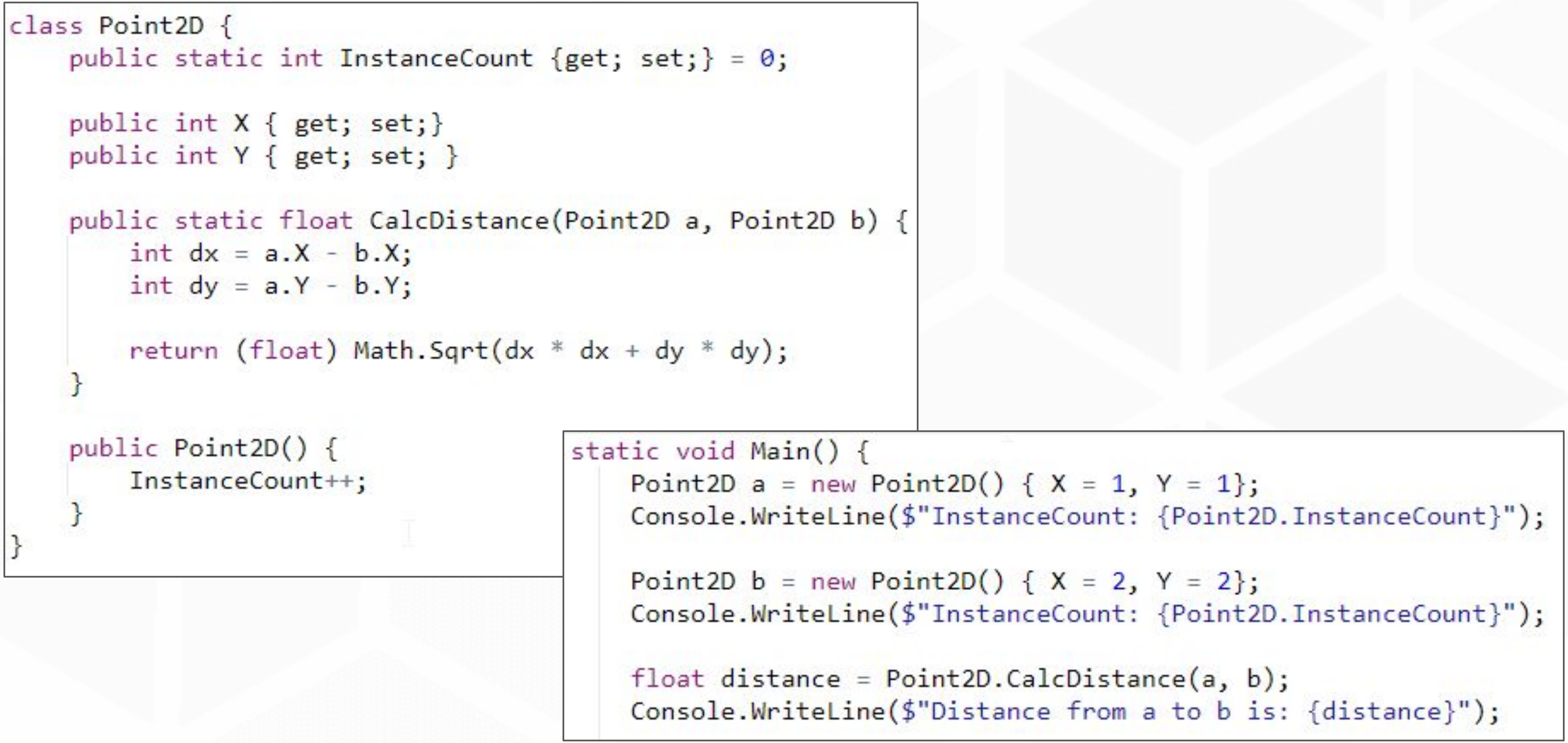
```
class Employee {
    public string FirstName {get; set;}
    public string LastName {get; set;}
    public string Tel {get; set;}

    public string FullName => $"{FirstName} {LastName}";
}
```

Static function



```
class Point2D {
    public int X { get; set;}
    public int Y { get; set; }

    public static float CalcDistance(Point2D a, Point2D b) {
        int dx = a.X - b.X;
        int dy = a.Y - b.Y;

        return Math.Sqrt(dx * dx + dy * dy);
    }
}
```

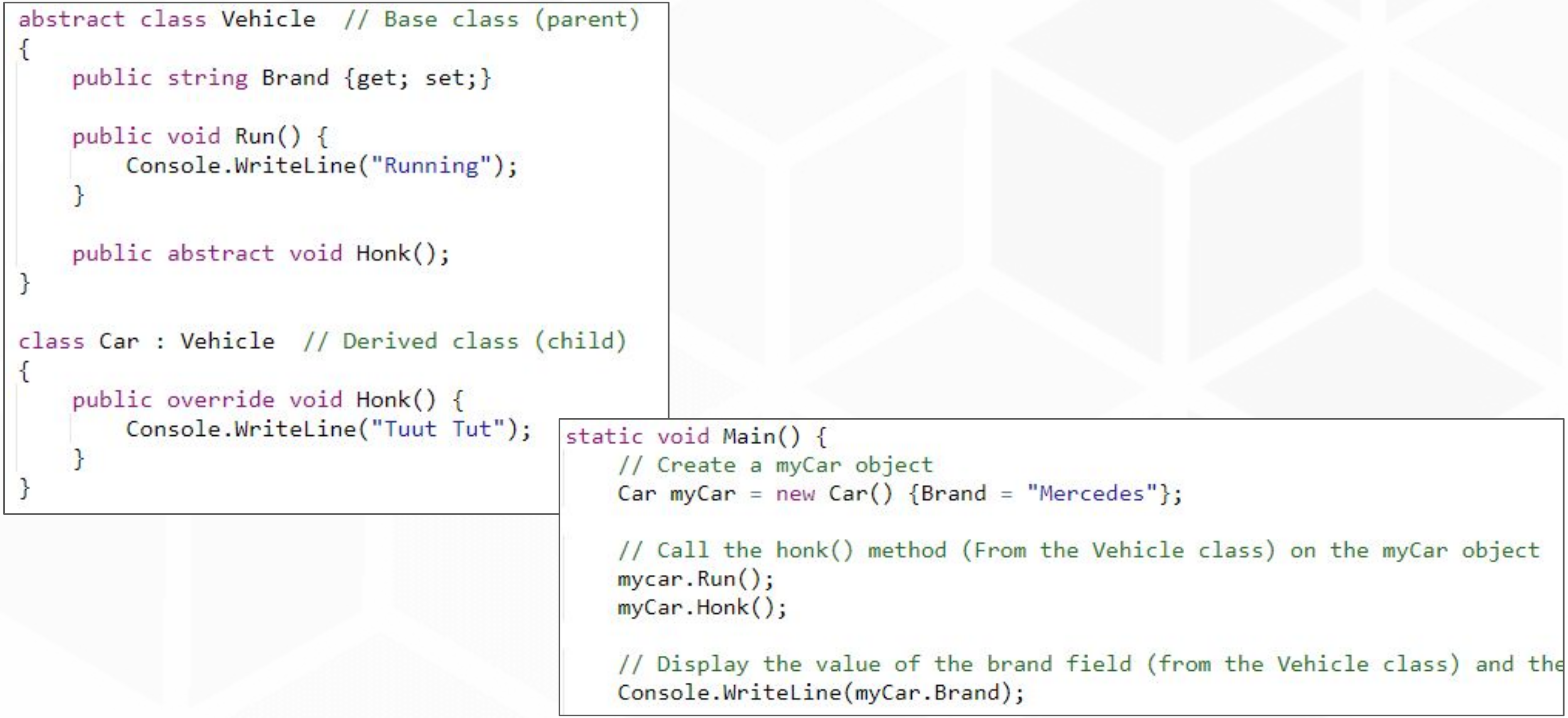
Static attribute / Property



```
class Point2D {
    public static int InstanceCount {get; set;} = 0;

    public int X { get; set;}
    public int Y { get; set; }

    public static float CalcDistance(Point2D a, Point2D b) {
        int dx = a.X - b.X;
        int dy = a.Y - b.Y;

        return (float) Math.Sqrt(dx * dx + dy * dy);
    }

    public Point2D() {
        InstanceCount++;
    }
}
```

```
static void Main() {
    Point2D a = new Point2D() { X = 1, Y = 1};
    Console.WriteLine($"InstanceCount: {Point2D.InstanceCount}");

    Point2D b = new Point2D() { X = 2, Y = 2};
    Console.WriteLine($"InstanceCount: {Point2D.InstanceCount}");

    float distance = Point2D.CalcDistance(a, b);
    Console.WriteLine($"Distance from a to b is: {distance}");
```

Inheritance - Reuse & Upgrade



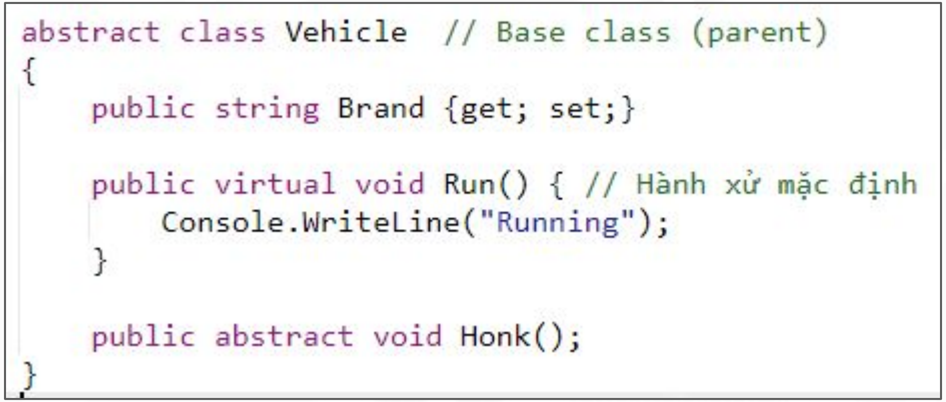
```
abstract class Vehicle  // Base class (parent)
{
    public string Brand {get; set;}

    public void Run() {
        Console.WriteLine("Running");
    }

    public abstract void Honk();
}

class Car : Vehicle  // Derived class (child)
{
    public override void Honk() {
        Console.WriteLine("Tuut Tut");
    }
}
```

```
static void Main() {
    // Create a myCar object
    Car myCar = new Car() {Brand = "Mercedes"};

    // Call the honk() method (From the Vehicle class) on the myCar object
    mycar.Run();
    myCar.Honk();

    // Display the value of the brand field (from the Vehicle class) and the
    Console.WriteLine(myCar.Brand);
```

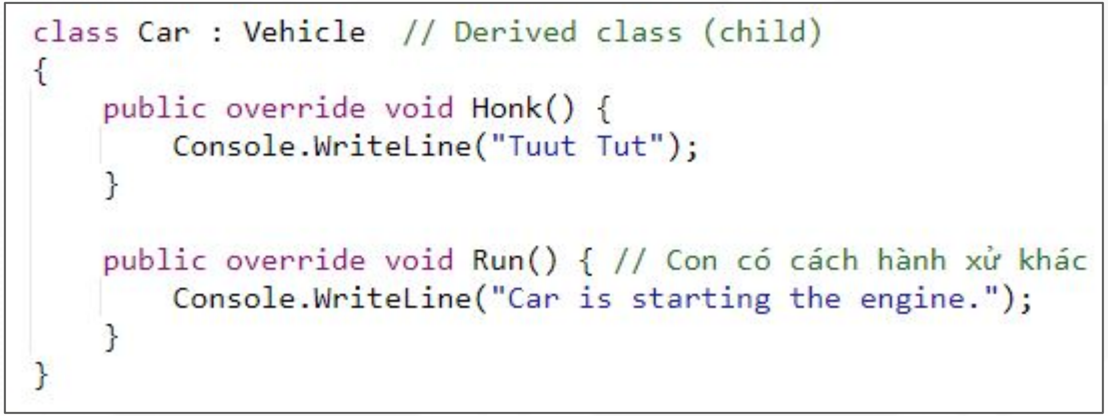
Examples of inheritance - Reuse & Upgrade

1. Point3D from Point2D
2. Person > Student > Teacher
3. Employee > DailyEmployee > ProductEmployee > Manager
4. Shape > Rectangle > Square > Circle > Ellipse
5. Character > Knight > Swordman > Pikeman

Overriding

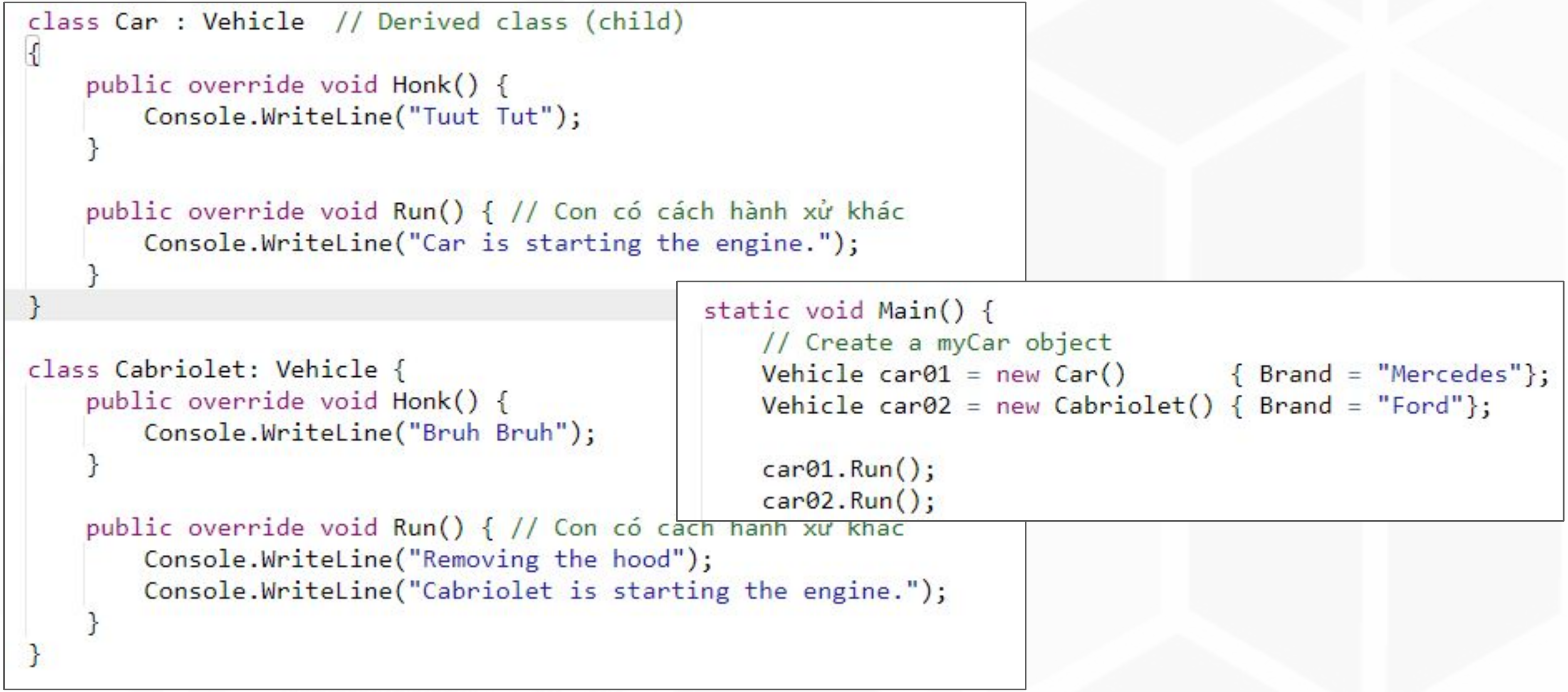


```
abstract class Vehicle  // Base class (parent)
{
    public string Brand {get; set;}

    public virtual void Run() { // Hành xử mặc định
        Console.WriteLine("Running");
    }

    public abstract void Honk();
}
```



```
class Car : Vehicle  // Derived class (child)
{
    public override void Honk() {
        Console.WriteLine("Tuut Tut");
    }

    public override void Run() { // Con có cách hành xử khác
        Console.WriteLine("Car is starting the engine.");
    }
}
```

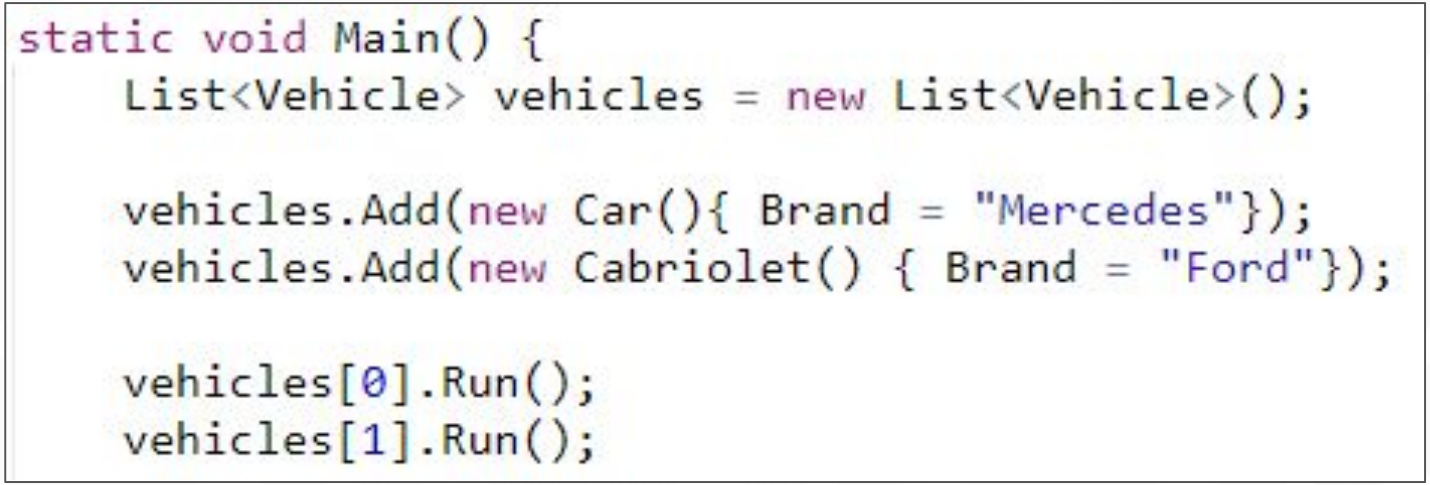
Polymorphism



```
class Car : Vehicle  // Derived class (child)
{
    public override void Honk() {
        Console.WriteLine("Tuut Tut");
    }

    public override void Run() { // Con có cách hành xử khác
        Console.WriteLine("Car is starting the engine.");
    }
}

class Cabriolet: Vehicle {
    public override void Honk() {
        Console.WriteLine("Bruh Bruh");
    }

    public override void Run() { // Con có cách hành xử khác
        Console.WriteLine("Removing the hood");
        Console.WriteLine("Cabriolet is starting the engine.");
    }
}
```

```
static void Main() {
    // Create a myCar object
    Vehicle car01 = new Car()       { Brand = "Mercedes"};
    Vehicle car02 = new Cabriolet() { Brand = "Ford"};

    car01.Run();
    car02.Run();
```

Array of objects



```
static void Main() {
    List<Vehicle> vehicles = new List<Vehicle>();

    vehicles.Add(new Car(){ Brand = "Mercedes"});
    vehicles.Add(new Cabriolet() { Brand = "Ford"});

    vehicles[0].Run();
    vehicles[1].Run();
```

Multiple inheritance

- ❑ C# does not support multiple classes inheritance
- ❑ But we have multiple interfaces inheritance

In-class exercise

A student will have this information: StudentID, FullName, GPA, Address

❑ Generate random 10 students

❑ Print out the average GPA of all the students

❑ Print out full information of the top 3 students with the highest GPA

Name the project: RandomStudents, submit back the onlinedgb / VS project link.

Instructions (extra work)

- StudentID: Length is 8, first two numbers: year 18/19/20 => Next 6 digits is random (0-9)
- Fullname: Firstname (Trần/Nguyễn/Lê/Lý/Đỗ/Đặng)
  - MiddleName: Tấn / Việt / Đức / Nhật / Khắc
  - Name: Quang / Minh / Thu / Thủy / Thắng
- GPA: random 0-10
- Address: Number (1-100), street name (Nguyễn Đình Chiểu, Hoàng Hoa Thám, CMT8, Quang Trung, Nguyễn Huệ), Ward (1-10), District (1-10/BinhTan/TanPhu)

References

https://www.w3schools.com/cs/cs_oop.asp

Parsable (.Net 7+)

Parse(String, IFormatProvider)

TryParse(String, IFormatProvider, TSelf)

Static polymorphism

https://learn.microsoft.com/en-us/dotnet/api/system.iparsable-1?view=net-7.0

Extension method

https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/extension-methods