Object-oriented Programming in C#

Understanding Object-orientation in C#

Gill Cleeren
CTO Xpirit Belgium

@gillcleeren

Overview

Understanding OOP in C#

Working with classes and objects

The pillars of OO

A word of warning!

This will be a more theoretical module!

Demos will start in the next module!

Some C# knowledge is required!

Basic C# syntax is assumed

Take a look at
“C# Fundamentals” if needed

Version Check

This version was created by using:

- C# 12
- Visual Studio 2022

Version Check

This course is 100% applicable to:

- C# 10, 11 & 12
- Visual Studio 2022 (any edition)

Most code will work fine with older versions of C#

Understanding OOP in C#

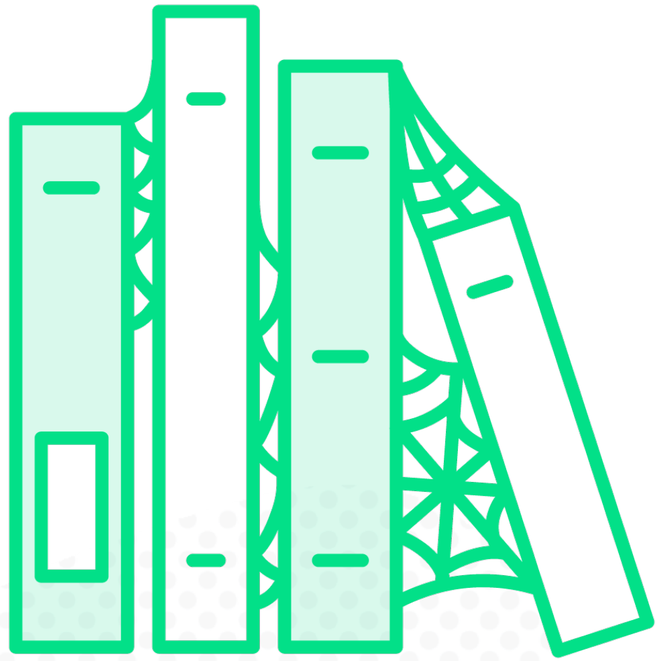
A Short History of OOP

Object-oriented programming

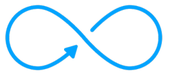
Over 50 years old

Moved from C++ and then into Java

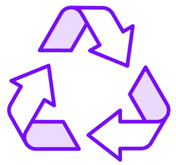
C# is Microsoft's answer



Understanding OOP

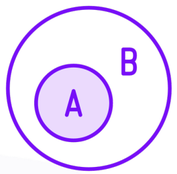


Objects interact with each other



Code reuse and easier to maintain

Aimed at all types of problems, including complex and large systems



Helps with breaking down the problem into smaller parts

OOP Languages

C++

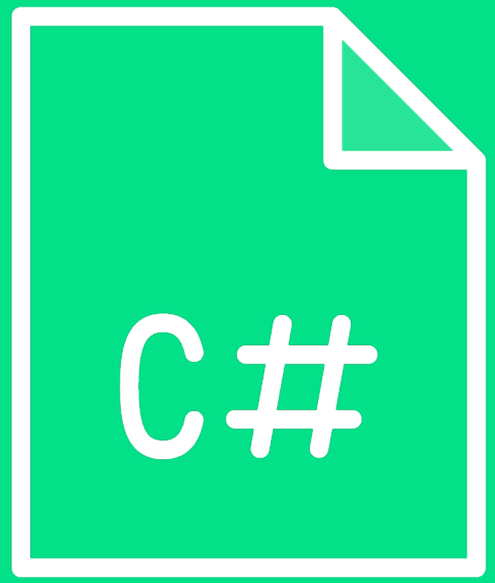
Java

Python

Ruby

JavaScript...

C#



C# and OOP

Object-oriented
Type-safe
Reusable code
Frequent updates
Modern tools

... and lots of fellow C# developers!

C# Building Blocks for Object-oriented Programming

Classes

Objects

Methods

Properties

Interfaces

Working with Classes and Objects

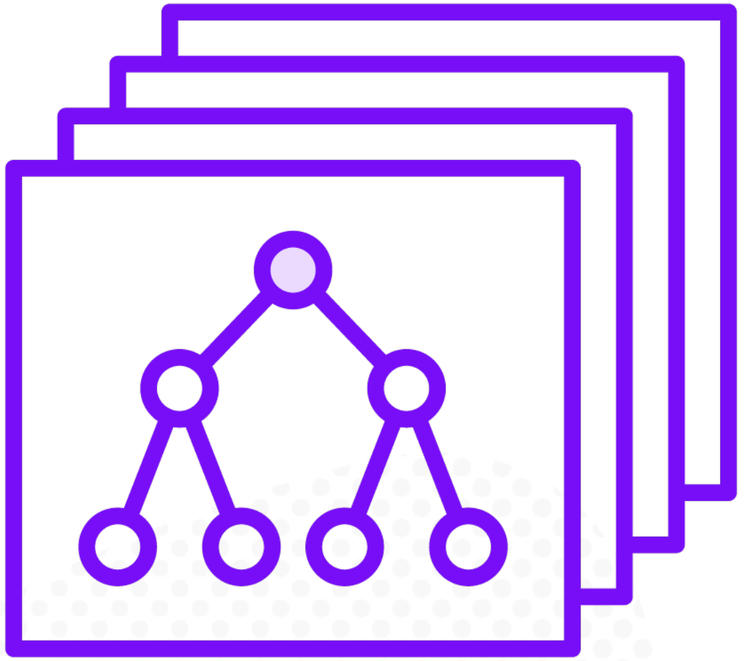
Understanding Classes

Concepts of the real world

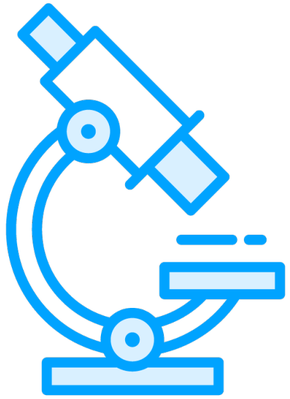
Used to categorize “types”

Building block of OOP

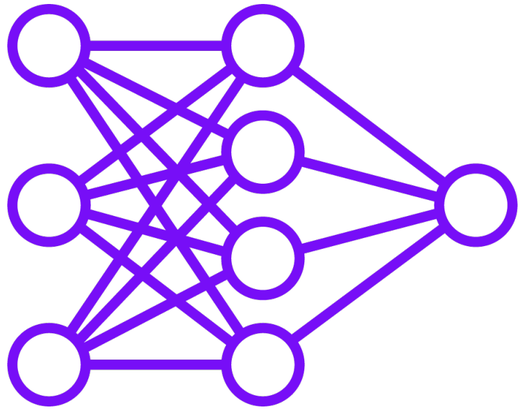
Reference type (like interfaces and delegates)



In C#, Everything is a Class...



Small programs
Main() or top-level Program.cs



Classes which will be instantiated
Create one or more objects

Classes and Objects

The Dog Class

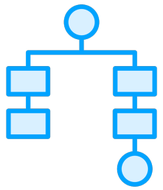
Hillary, “a” dog

Violet, “a” dog

Bob, “a” dog

Classes in C#

Blueprint of an object



Defines data and functionality to work on its data

Created using class keyword

The Class Template

```
public class MyClass
{
    private int counter;

    public void MyMethod()
    {
        //Do something great here
    }
}
```

Contents of a Class

Fields

Methods

Properties

Events

Exploring a Simple Class

```
public class Employee
{
    public Employee()
    {
        //Initialization code goes here
    }

    private string firstName;
    private int age;

    public string FirstName
    {
        get { return firstName; }
        set { firstName = value; }
    }

    public void PerformWork()
    {
        //method implementation goes here
    }
}
```

Class Declaration and Naming

```
public class Employee
{
    public Employee()
    {
        //Initialization code goes here
    }

    private string firstName;
    private int age;

    public string FirstName
    {
        get { return firstName; }
        set { firstName = value; }
    }

    public void PerformWork()
    {
        //method implementation goes here
    }
}
```

Fields Contain Data

```
public class Employee
{
    public Employee()
    {
        //Initialization code goes here
    }

    private string firstName;
    private int age;

    public string FirstName
    {
        get { return firstName; }
        set { firstName = value; }
    }

    public void PerformWork()
    {
        //method implementation goes here
    }
}
```

Properties Control Access to Data

```
public class Employee
{
    public Employee()
    {
        //Initialization code goes here
    }

    private string firstName;
    private int age;

    public string FirstName
    {
        get { return firstName; }
        set { firstName = value; }
    }

    public void PerformWork()
    {
        //method implementation goes here
    }
}
```

Methods Define Functionality

```
public class Employee
{
    public Employee()
    {
        //Initialization code goes here
    }

    private string firstName;
    private int age;

    public string FirstName
    {
        get { return firstName; }
        set { firstName = value; }
    }

    public void PerformWork()
    {
        //method implementation goes here
    }
}
```

Access Modifiers Define Access to Members

```
public class Employee
{
    public Employee()
    {
        //Initialization code goes here
    }

    private string firstName;
    private int age;

    public string FirstName
    {
        get { return firstName; }
        set { firstName = value; }
    }

    public void PerformWork()
    {
        //method implementation goes here
    }
}
```

What are members?

Classes contain members

Access modifiers

Constructors Are Used for Initialization

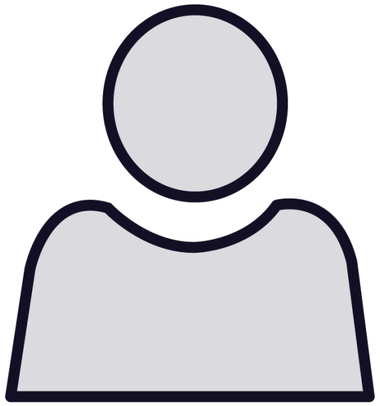
```
public class Employee
{
    public Employee()
    {
        //Initialization code goes here
    }

    private string firstName;
    private int age;

    public string FirstName
    {
        get { return firstName; }
        set { firstName = value; }
    }

    public void PerformWork()
    {
        //method implementation goes here
    }
}
```

Classes and Objects



Employee Class

Age is a field

Object 1
FirstName: John
Age: 27

Object 2
FirstName: Mary
Age: 31

Object 3
FirstName: George
Age: 22

```
Employee employee = new Employee();
```

Instantiating an Object

The new operator

```
employee.PerformWork();

employee.PerformWork(10);

employee.FirstName = "Gill";
```

Working with an Object

Invoke methods, properties...

Types in the CTS

Class

Enumeration

Struct

Interface

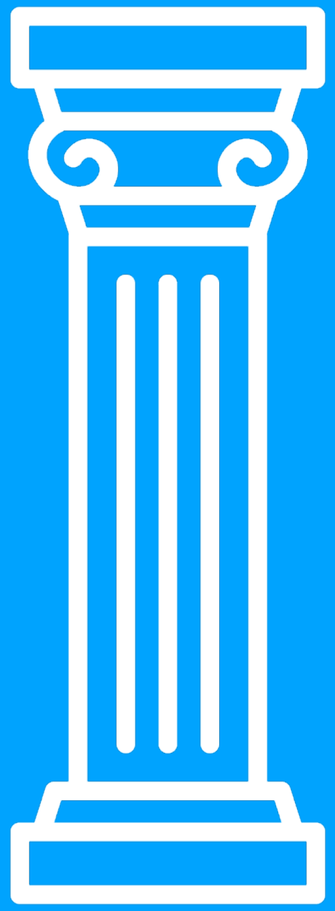
Delegate

Record

More on Classes and Objects

We will keep using classes and objects
and explore all we can do with them applying
OO in C#!

The Principles of Object-oriented Design



Introducing the 4 Pillars

Help with writing clean, object-oriented code

Covered by C# language features

The Pillars of OOP

Abstraction

Encapsulation

Inheritance

Polymorphism

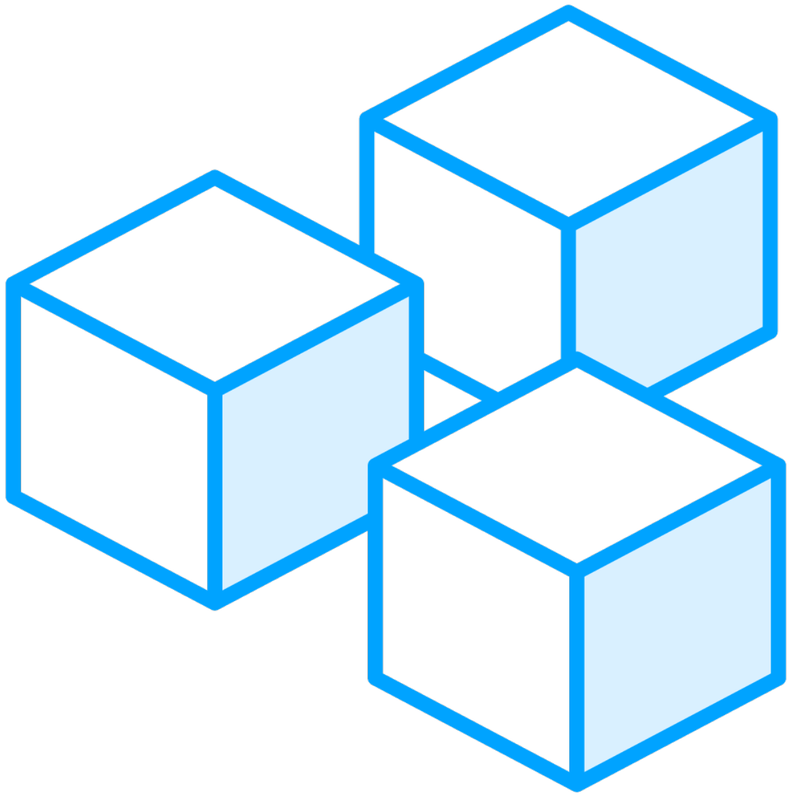
The Pillars of OOP

Abstraction

Encapsulation

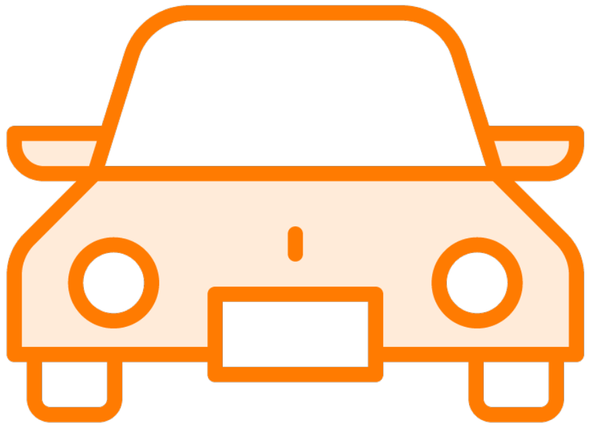
Inheritance

Polymorphism



The concept of abstraction

- Think about the essential concepts, not the background details
- Create layer of abstraction
- Expose simple handles to interact without knowing about the details
- Focus on what it does, not how it does it



Abstraction applied on Car

- Steering wheel, brake pedal expose simple interface to the user
  - This is the abstraction
- No need to know how the engine, brakes really work
  - Only necessary aspects are exposed

Benefits of Applying Abstraction

Reduce complexity, only simple interfaces

Hide and secure important functionality

Better for maintenance

Updates don’t break existing interface mostly

Thinking in Abstraction

Employee

Name
Age
Weight
Address
Hair color
Perform work

The Pillars of OOP

Abstraction

Encapsulation

Inheritance

Polymorphism

The Pillars of OOP

Abstraction

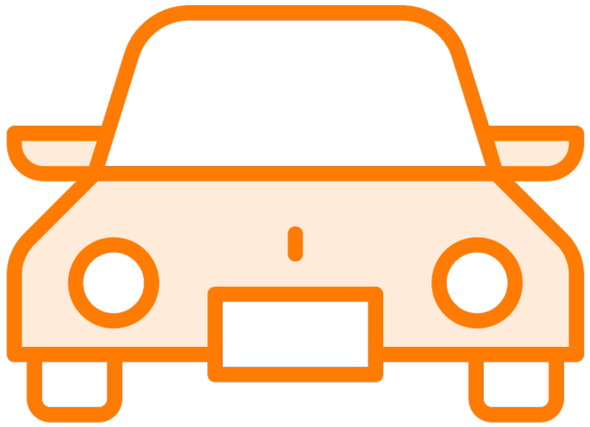
Encapsulation

Inheritance

Polymorphism

Encapsulation

- All data and functionality on this data is encapsulated inside object
- Only certain information is exposed
- Data is hidden inside object
- Use access modifiers



Encapsulation applied on a Car

- Defines speed, engine, drive(), speedUp()...
- Bound together in one unit
- Engine data must be hidden (private)
- speedUp() can be working on data but still be publicly available

```
public class Car
{

    private int id;

    private int temperature;

    public int Id
    {
        get { return id; }
        set {
                if(id > 0)
                    id = value;
            }
    }
}
```

- Public part is useable from outside
- Private part is only useable from within the class itself
- Important data can be hidden so it can’t be changed (intentionally or by mistake)
- We can control how the data changes

The Pillars of OOP

Abstraction

Encapsulation

Inheritance

Polymorphism

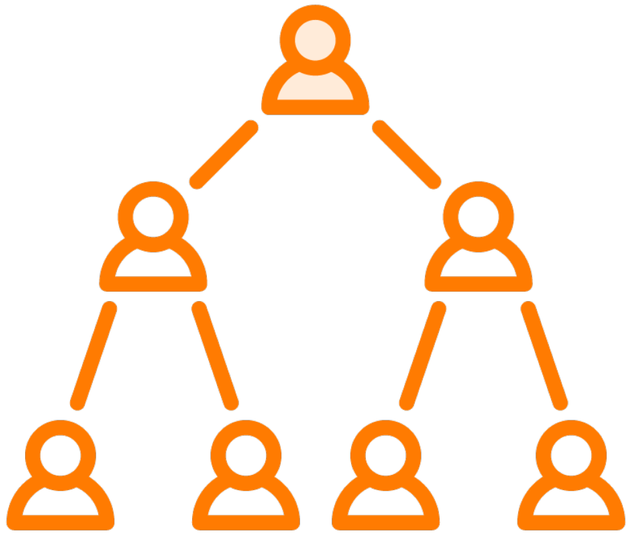
The Pillars of OOP

Abstraction

Encapsulation

Inheritance

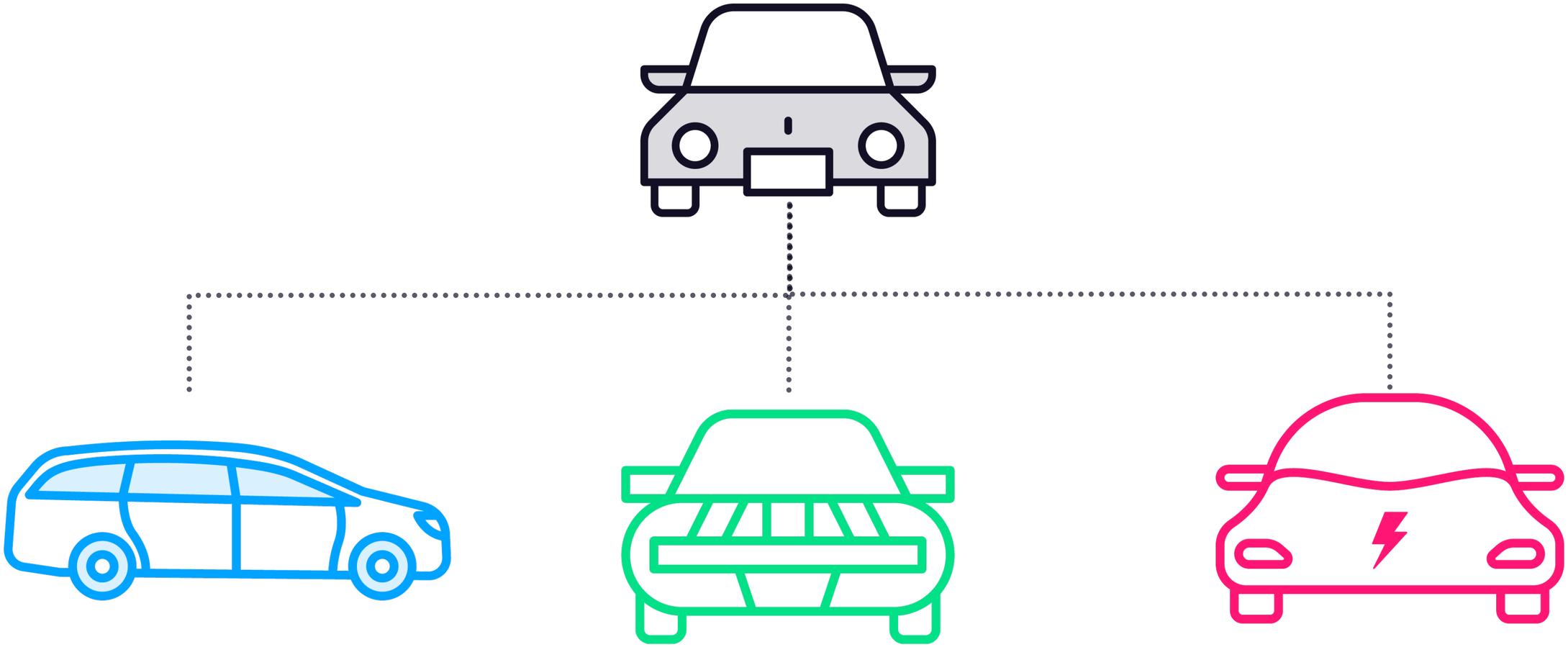
Polymorphism



Inheritance

- Inherit features from other classes
- Create hierarchy
  - Is-A relation
- Improve reuse of code between parent and child class
- Child class can extend functionality and attributes

Inheritance Applied on a Car



```
public class Car
{
    private int maxSpeed;

    public int MaxSpeed { get => maxSpeed; set => maxSpeed = value; }

    public void Drive()
    {
    }
}
```

The Parent Car Class

```
public class SportsCar : Car
{
    private bool isRoofOpen;
}
```

The SportsCar Class

The Pillars of OOP

Abstraction

Encapsulation

Inheritance

Polymorphism

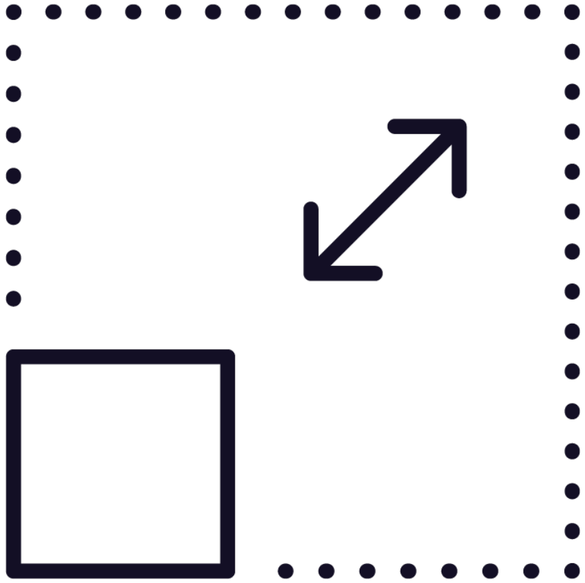
The Pillars of OOP

Abstraction

Encapsulation

Inheritance

Polymorphism



Polymorphism

- Allow methods to execute differently
- virtual and override keywords in C#
- On objects of different types, we can invoke the same methods

Applying Method Overriding

```
public class Car
{
    public virtual void Drive()
    { }
}

public class ElectricCar : Car
{
    public override void Drive()
    {
        base.Drive();
    }
}
```

Summary

C# supports object-oriented programming

OOP design principles

- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

Up Next:

Designing an Object-oriented Solution from a Business Case