C# INTERMEDIATE

Fields

Agenda

- Initialization
- Read-only fields

Initialization

```
public class Customer
{
    List<Order> Orders;

    public Customer()
    {
        Orders = new List<Order>();
    }
}
```

Initialization

```
public class Customer
{
    List<Order> Orders = new List<Order>();
}
```

Read-only Fields

```
public class Customer
{
    readonly List<Order> Orders = new List<Order>();
}
```

Access Modifiers

- Public
- Private
- Protected
- Internal
- Protected Internal

What?

A way to control access to a class and/or its members.

Why?

To create safety in our programs.

How?

```
public class Customer
{
    private string Name;
}


…
var john = new Customer();
john.Name; // won't compile
```

Object-oriented programming

- Encapsulation / Information Hiding
- Inheritance
- Polymorphism

Encapsulation (in practice)

- Define fields as private
- Provide getter/setter methods as public

```
public class Person
{
    private string Name;

    public void SetName(string name)
    {
        if (!String.IsNullOrEmpty(name))
            this.Name = name;
    }

    public string GetName()
    {
        return Name;
    }
}
```

```
public class Person
{
    private string _name;

    public void SetName(string name)
    {
        if (!String.IsNullOrEmpty(name))
            this._name = name;
    }

    public string GetName()
    {
        return _name;
    }
}
```

Program.cs

AccessModifiers.Person

GetBirthdate()

```
using System;

namespace AccessModifiers
{
    public class Person
    {
        private DateTime _birthdate;

        public void SetBirthdate(DateTime birthdate)
        {
            _birthdate = birthdate;
        }

        public DateTime GetBirthdate()
        {
            return _birthdate;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            var person = new Person();
            person.SetBirthdate(new DateTime(1982, 1, 1));
            Console.WriteLine(person.GetBirthdate());
```

C# INTERMEDIATE

Properties

What?

A class member that encapsulates a getter/setter for accessing a field.

Why?

To create a getter/setter with less code.

```
public class Person
{
    private DateTime _birthdate;

    public void SetBirthdate(DateTime birthdate)
    {
        this._birthdate = birthdate;
    }

    public DateTime GetBirthdate()
    {
        return _birthdate;
    }
}
```

How?

```
public class Person
{
    private DateTime _birthdate;

    public DateTime Birthdate
    {
        get { return _birthdate; }
        set { _birthdate = value; }
    }
}
```

Auto-implemented Properties

```
public class Person
{
    public DateTime Birthdate { get; set; }
}
```