Types, Objects and OOP

Simon Robinson

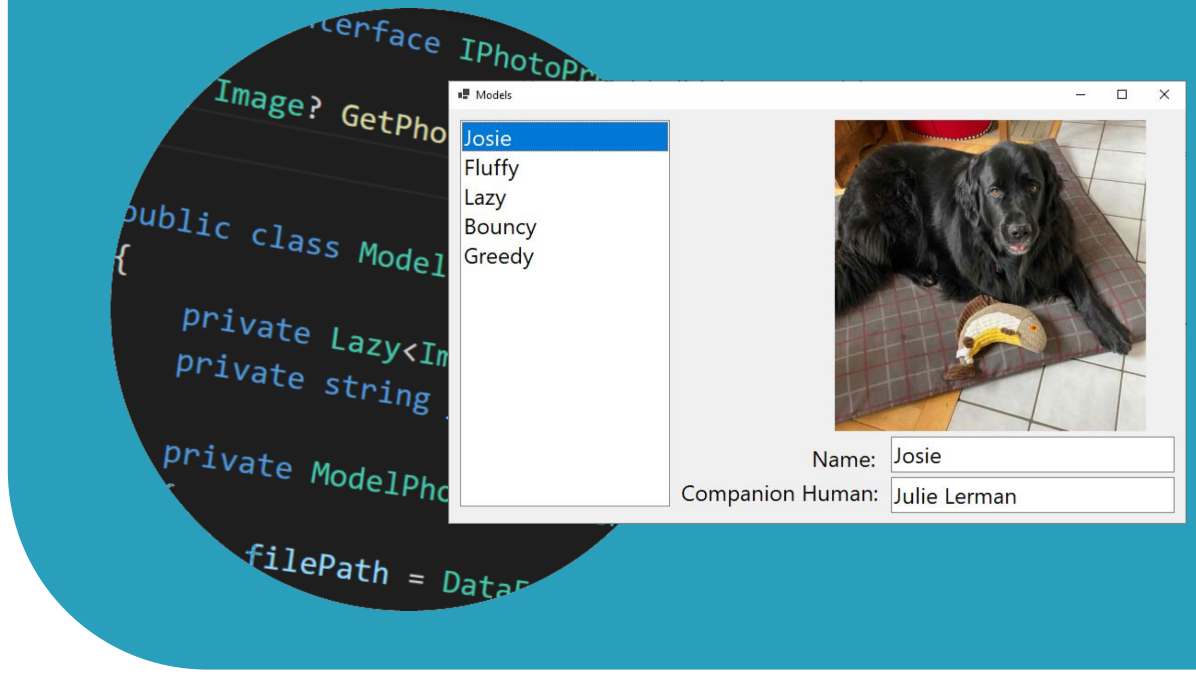
Software Developer

@TechieSimon www.SimonRobinson.com

Module Structure

First part:

Main demo: OOP techniques



```
terface IPhotoPr
Image? GetPho

public class Model
{
    private Lazy<Im
    private string

    private ModelPho

    filePath = DataF
```

Josie
Fluffy
Lazy
Bouncy
Greedy

Name: Josie

Companion Human: Julie Lerman

Second part:

Overview

Keeping your data valid

- Initializing fields up a hierarchy

Minimizing resource usage

- Lazy load objects
- Implement singleton class

Accessing private members

- Can sometimes actually help encapsulation!

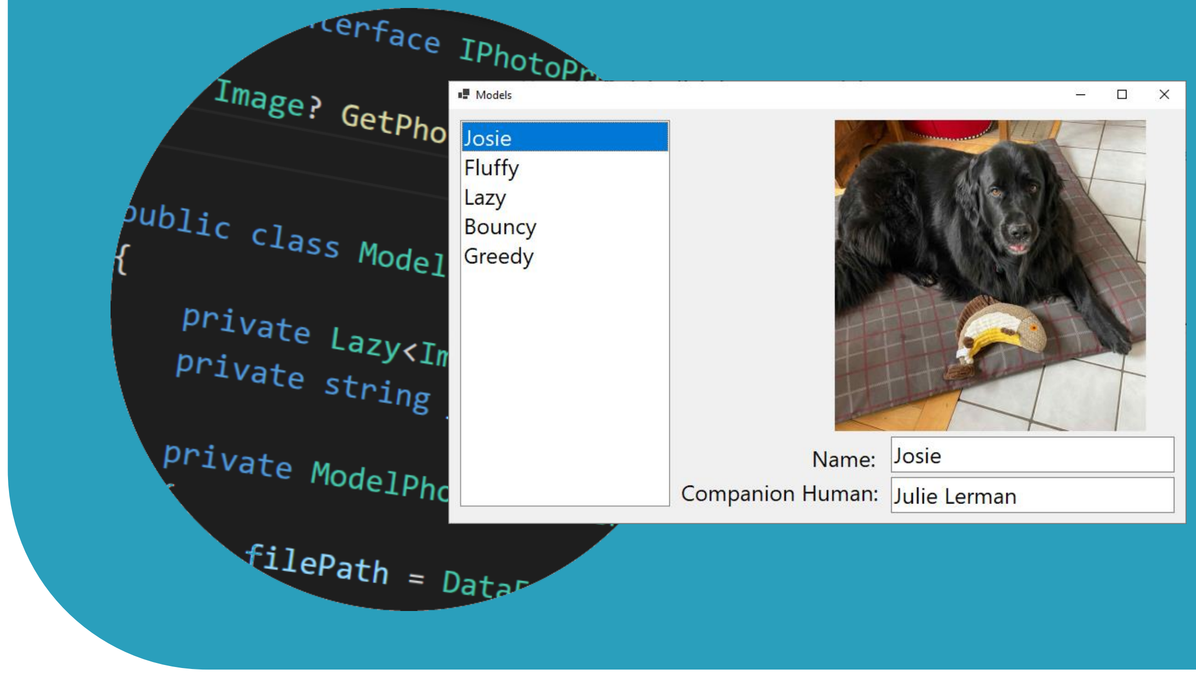
Safely casting objects using pattern matching

- Recent C# feature

Module Structure

First part:

Main demo: Solving OOP problems



```
terface IPhotoPr
Image? GetPho

public class Model
{
  private Lazy<Im
  private string

  private ModelPho

    filePath = DataF
```

Models

Josie
Fluffy
Lazy
Bouncy
Greedy

Name: Josie

Companion Human: Julie Lerman

Second part:

Choosing types

```
struct
class
record struct
record class
```

You can watch either part separately

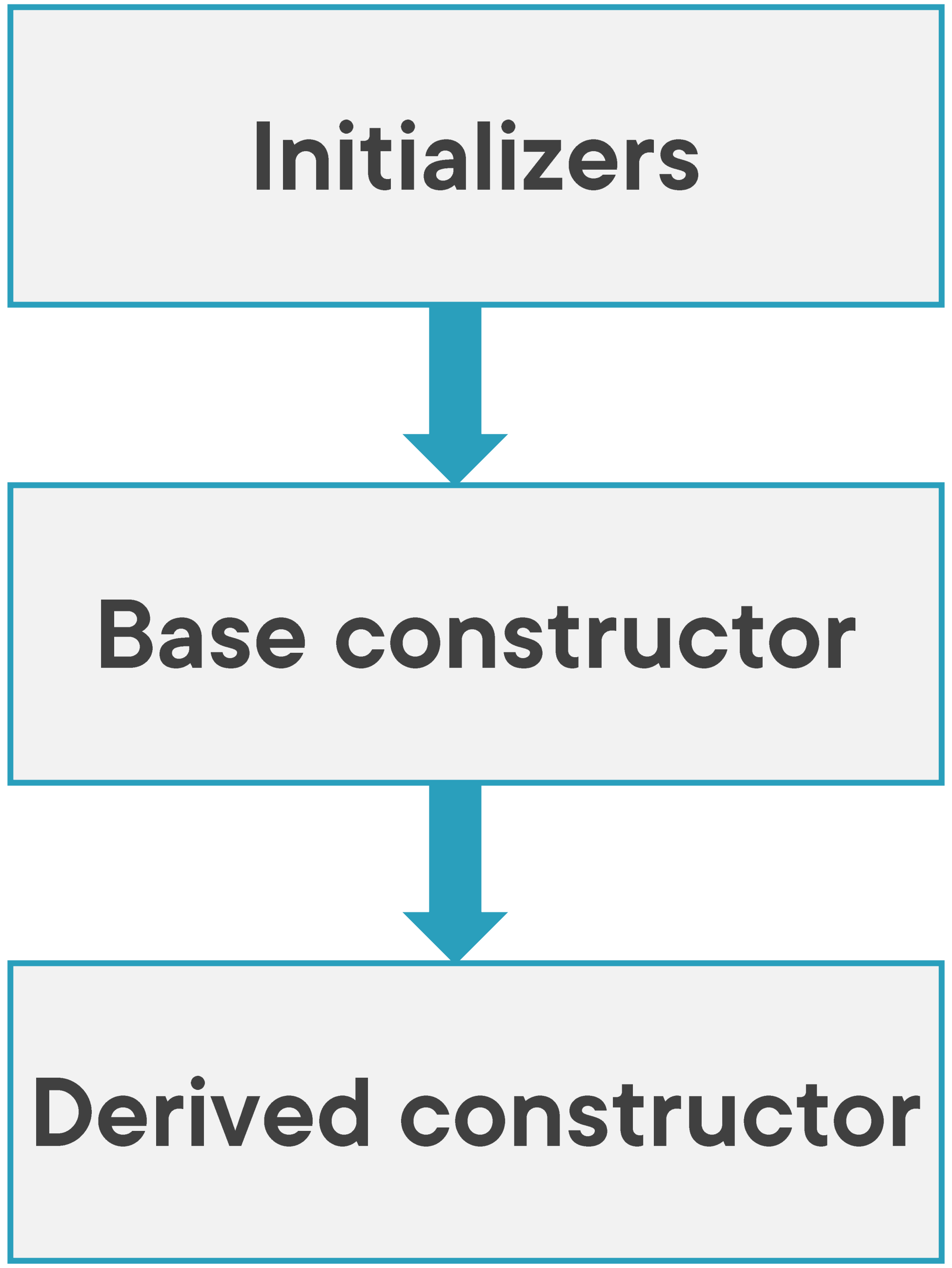
Initializing up an Inheritance Chain

Demo

Modelling agency app

- Base and derived classes for models
- Examine how these classes ensure fields are initialized up the inheritance chain

Initialization



Remember the initialization order

You can use data from items that have already run:

- Initializers can use only constants and statics

- Base class constructors can use initializer data

- Derived class constructors can use all base class data

Lazy Loading

Lazy Loading

Avoiding instantiating an object until it’s needed

You don’t waste resources loading an object that might not be required

Demo

Code to load and provide photos

- Modify to lazy-load photos
  - Coding the logic yourself
  - Using Lazy<T>

Writing Singletons

Singleton Classes

Only one instance of a singleton class can exist

Singleton classes allow conserving resources

Demo

Provide default photo

- Singleton to ensure this photo is only loaded once

Accessing Private Members of a Type

Accessing Private Members

```
class MyClass1
{
    private int _myField;
    // etc.
}
```

```
class MyClass2
{
    // etc.
```

Why would you do that?

Sometimes specific classes work
so closely together…

… that it can actually help encapsulation

Demo

Implement a factory

- The factory will require access to private members
- Solve with a nested class

Conditional Logic with Type Pattern Matching

Demo

Examine code to display images

- Code must take different actions for different types
- Pattern matching provides the solution

Deciding between Struct, Class, Record Struct and Record Class

	Classes	Structs
Inheritance	✓	✗

```
public class MyClass : MyBaseClass
{
```

You can only inherit from classes

Structs do not support inheritance

	Classes	Structs
Inheritance	✓	✗
Interfaces	✓	⚠

Interfaces are treated as reference types

	Classes	Structs
Inheritance	✓	✗
Interfaces	✓	⚠
Data validation	✓	⚠

Validation code would normally go in the constructor

But with structs, there are a few situations where no constructor will run

	Classes	Structs
Inheritance	✓	✗
Interfaces	✓	⚠
Data validation	✓	⚠
Performance (lifetime)	⚠	✓

Instantiation and lifetime management are more expensive for classes

- That’s because they must be allocated on the managed heap, and garbage-collected

Structs don’t have that overhead

	Classes	Structs
Inheritance	✓	✗
Interfaces	✓	⚠
Data validation	✓	⚠
Performance (lifetime)	⚠	✓
Performance (as arguments)	✓	✓/⚠

Classes pass references, which is quick

Passing structs involves copying values

You can pass structs by reference
by explicitly specifying
reference arguments

	Classes	Structs
Inheritance	✓	✗
Interfaces	✓	⚠
Data validation	✓	⚠
Performance (lifetime)	⚠	✓
Performance (as arguments)	✓	✓/⚠
Immutability	⚠	✓

You can make any type immutable

But there's better language support for immutability for structs

	Classes	Structs
Inheritance	✓	✗
Interfaces	✓	⚠
Data validation	✓	⚠
Performance (lifetime)	⚠	✓
Performance (as arguments)	✓	✓/⚠
Immutability	⚠	✓
Reference equality	✓	✗
Value equality	⚠	⚠

Are different instances with the same value equal?

Value equality says yes

Reference equality says no

Value equality makes sense for things like business objects

Reference equality makes sense for things like controls

Only classes support reference equality

You can implement value equality for all types – but it takes work

	Classes	Structs
Inheritance	✓	✗
Interfaces	✓	⚠
Data validation	✓	⚠
Performance (lifetime)	⚠	✓
Performance (as arguments)	✓	✓/⚠
Immutability	⚠	✓
Reference equality	✓	✗
Value equality	⚠	⚠

✓ For records

Records

```
public record class MyRecord
{
```

```
public record struct MyValueRecord
{
```

For records, the compiler will implement value equality for you

Useful for types that are primarily groups of properties

	Classes	Structs
Inheritance	✓	✗
Interfaces	✓	⚠
Data validation	✓	⚠
Performance (lifetime)	⚠	✓
Performance (as arguments)	✓	✓/⚠
Immutability	⚠	✓
Reference equality	✓	✗
Value equality	⚠	⚠

✓ For records

Next up... some examples!

A challenge:

Which would you choose?

`record class`

`record struct`

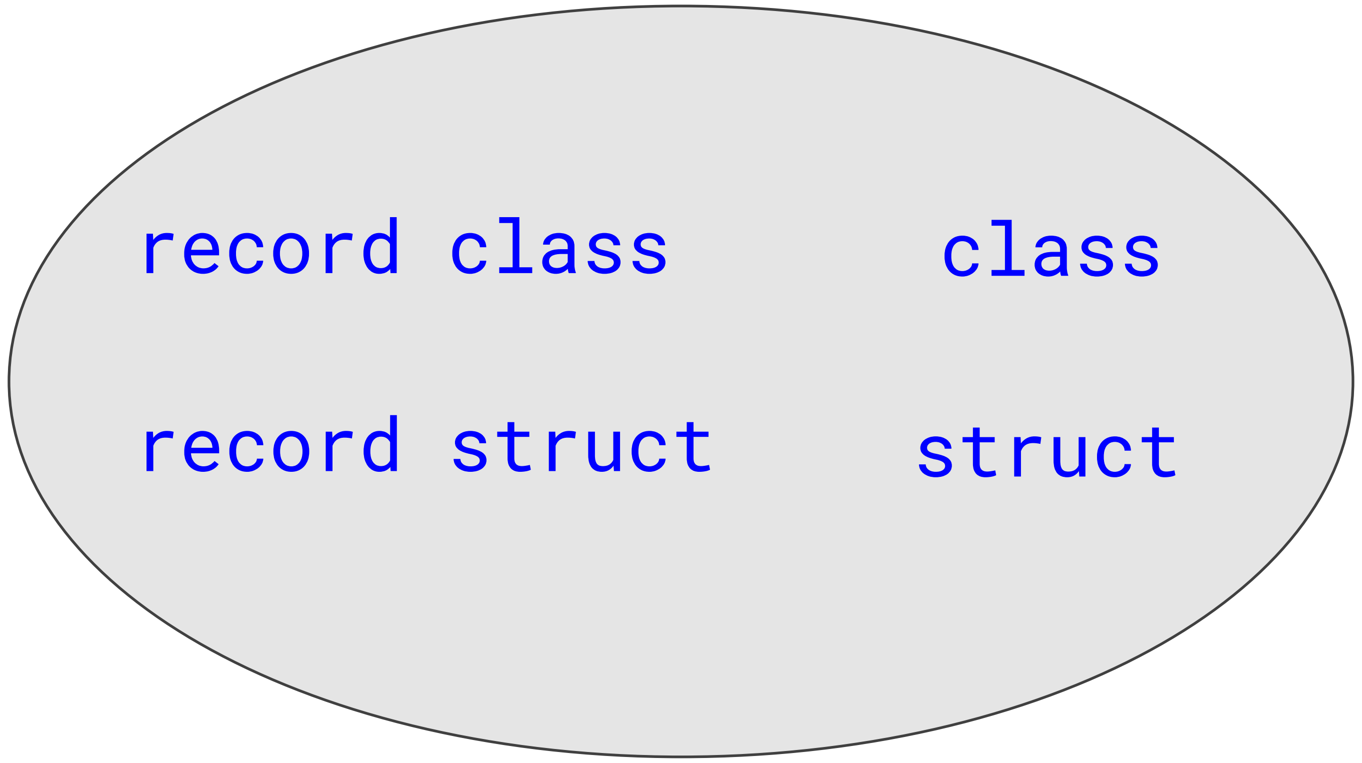
`struct`

`class`

Struct or Class? Customer Address

```
public                     CustomerAddress
{
```

```
public                     CustomerAddress
{
```



It has lots of fields

Structs are often more suited to small types where performance is important

You probably want data validation

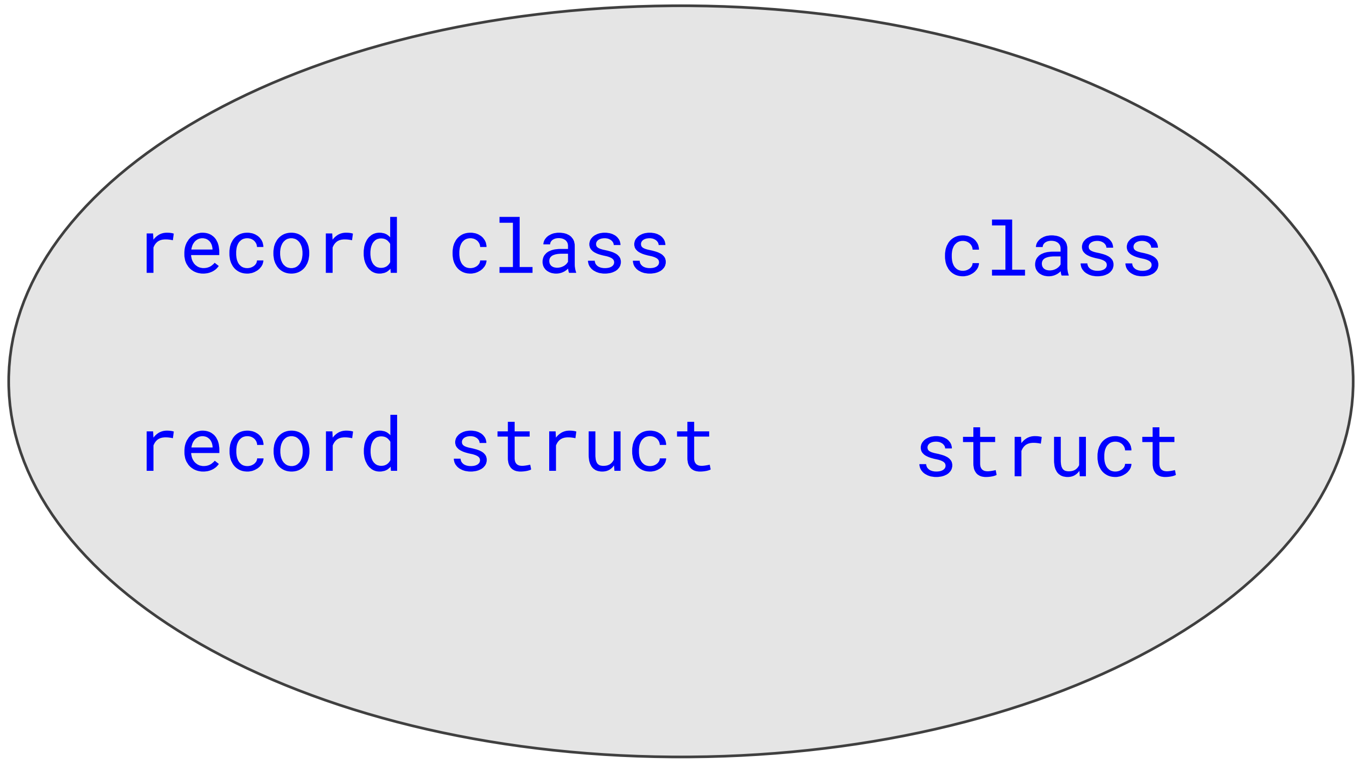
Could be record class because it’s largely a property bag

Struct or Class? Date-time Picker Control

Controls will need reference equality

It’s likely to be part of a class hierarchy

```
public                    DateTimePicker
{
```



record class

class

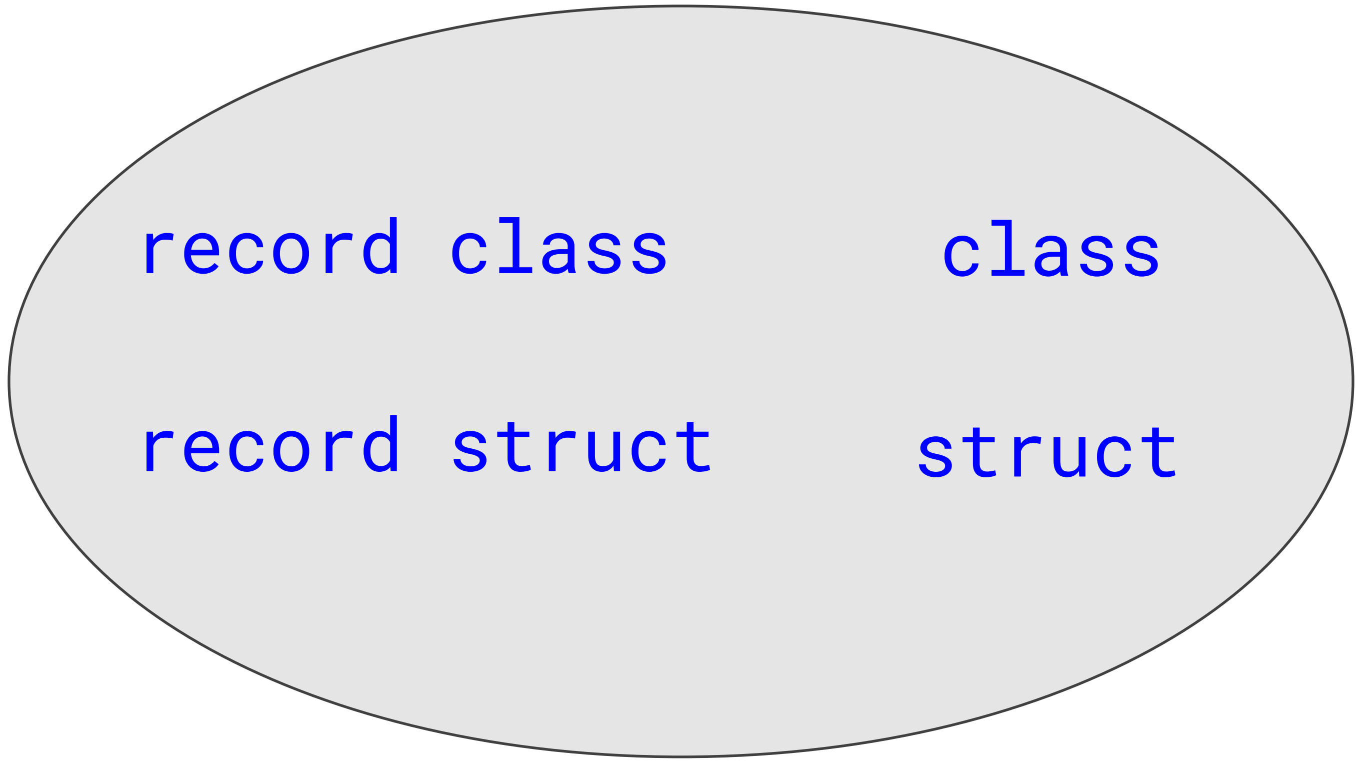
record struct

struct

Struct or Class? 3-dimensional Point

```
public                     Point3D
{
```

```
public                     Point3D
{
```



```
record class      class

record struct     struct
```

Small

May instantiate lots of instances in tight loops

So instantiation performance will be important

You want value, not reference, equality

Could make a record

Struct or Class? Data Repository

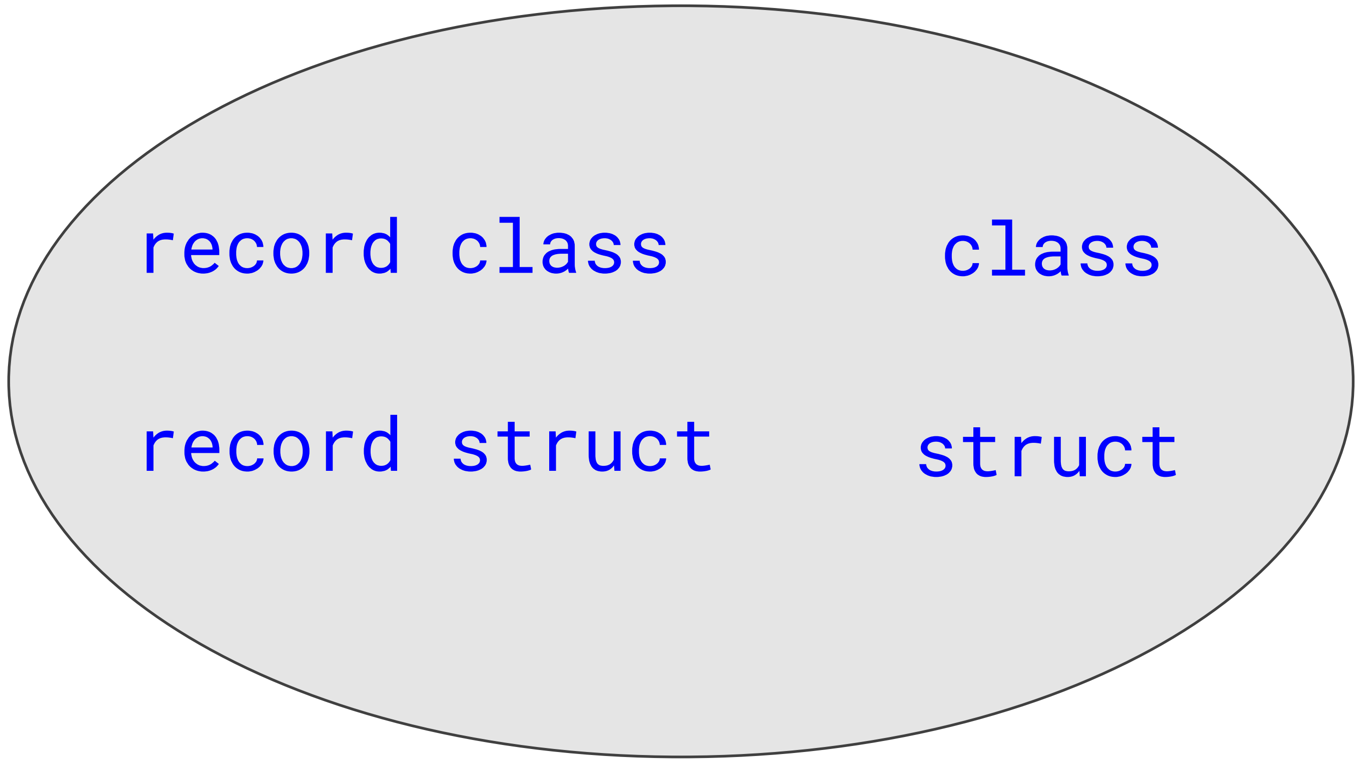
Likely to want to access through an interface

Interfaces favour classes

Instantiation costs not an issue - you won't create many instances!

Not a record: A repository largely provides methods – it's not a property bag

```
public                 DataRepository
{
```



record class

class

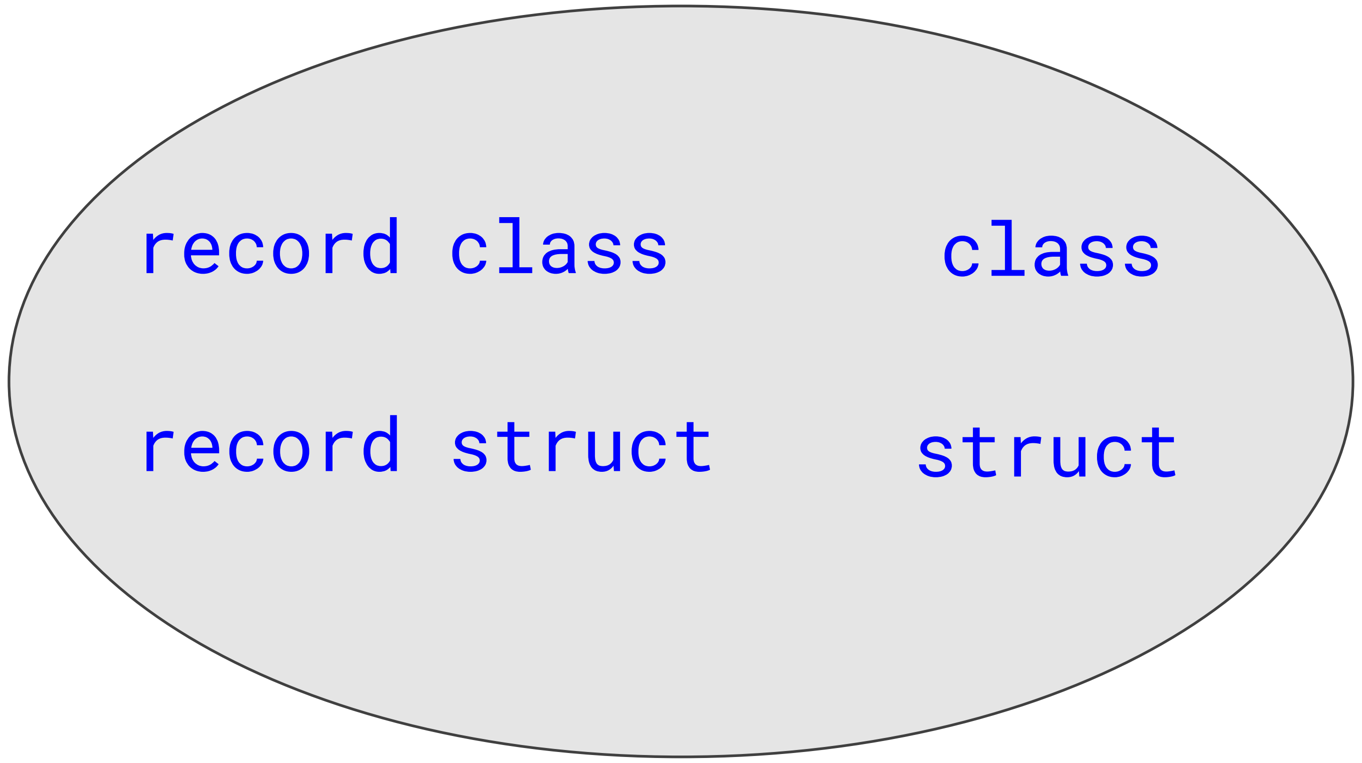
record struct

struct

Struct or Class? Product Base Type

```
public                    ProductBase
{
```

```
public                    ProductBase
{
```



record class
class
record struct
struct

You can’t use structs in inheritance hierarchies!

Record class or ordinary class?

- Depends how you want to use the derived Product types

You can’t mix records and non-records in inheritance hierarchies

- Which one you choose for ProductBase determines every derived Product type

Struct or Class?

```
Pluralsight.CShPlaybook
tualProduct(CanUseInTemplate = true)]
public class ComputerProduct : Product
{
    public ComputerProduct(string name, Produc
        ram, int processorCount)
        : base(name, status, price)
    {
        Ram = ram;
        ProcessorCount = processorCount;
    }

    public string Ram { get; }

    public int ProcessorCount { get; }
```

Most scenarios favour classes

Structs for small types where performance is important and you want value equality

Records typically for business objects

These are suggestions, not hard rules

Summary

Initializing up a hierarchy

- base( ) to invoke base constructors
- Derived constructors can rely on base types being fully initialized

Use resources efficiently

- Lazy<T> to lazy-load resources
- Singletons to ensure shared resources are only loaded once

Nested classes

- Give helper classes access to the classes they are helping
- Factory classes can use this

Summary

Type pattern matching

- Conditional logic based on types

Choosing class or struct

- Structs typically for frequently instantiated small types
- Otherwise classes
- Records for property bags with value equality