Herding Nulls

and other C# stories from the future

Mads Torgersen, C# lead designer

Microsoft

Chapter One

Herding nulls

Every reference type allows null!

Code must be defensive about it

C# shares this with most object oriented languages

Not all languages have this problem!

Differentiate between nullable and non-nullable

Use pattern matching to conditionally “unpack” nullables

1. ***Expression*** of intent

2. ***Enforcement*** of intent

Within an existing language!

```
public class Person
{
    public string FirstName  { get; set; }
    public string MiddleName { get; set; }
    public string LastName   { get; set; }
}
```

```
public class Person
{
    public string! FirstName  { get; set; }
    public string? MiddleName { get; set; }
    public string! LastName   { get; set; }
}
```

```
public class Person
{
    public string! FirstName  { get; set; }
    public string  MiddleName { get; set; }
    public string! LastName   { get; set; }
}
```

```
public class Person
{
    public string  FirstName  { get; set; }
    public string? MiddleName { get; set; }
    public string  LastName   { get; set; }
}
```

1. *Protect non-null types from nulls*
2. *Protect nulls from dereference*

Must be optional

Turn it on when you're ready to know

Can't affect semantics - warnings, not errors

Must do a good job with existing code

Can’t force you to rewrite good code

Recognize existing ways of ensuring null safety (checks and assignments)

Can’t be exhaustive

Tradeoff between completeness and convenience

Color illustration

Nullable reference types

Chapter Two

Async streams

Chapter Three

Extension everything

```
extension Student extends Person
{
    // static field
    static Dictionary<Person, Professor> enrollees = new Dictionary<Person, Professor>();

    // instance method
    public void Enroll(Professor supervisor) { enrollees[this] = supervisor; }

    // instance property
    public Professor? Supervisor => enrollees.TryGetValue(this, out var supervisor) ? supervisor : null;

    // static property
    public static ICollection<Person> Students => enrollees.Keys;

    // instance constructor
    public Person(string name, Professor supervisor) : this(name) { this.Enroll(supervisor); }
}
```

```
extension Student extends Person
{
    // static field
    static Dictionary<Person, Professor> enrollees = new Dictionary<Person, Professor>();

    // instance method
    public void Enroll(Professor supervisor) { enrollees[this] = supervisor; }

    // instance property
    public Professor? Supervisor => enrollees.TryGetValue(this, out var supervisor) ? supervisor : null;

    // static property
    public static ICollection<Person> Students => enrollees.Keys;

    // instance constructor
    public Person(string name, Professor supervisor) : this(name) { this.Enroll(supervisor); }
}

Person mads = new Person("Mads Torgersen", tonyHoare);
```

```
extension Student extends Person
{
    // static field
    static Dictionary<Person, Professor> enrollees = new Dictionary<Person, Professor>();

    // instance method
    public void Enroll(Professor supervisor) { enrollees[this] = supervisor; }

    // instance property
    public Professor? Supervisor => enrollees.TryGetValue(this, out var supervisor) ? supervisor : null;

    // static property
    public static ICollection<Person> Students => enrollees.Keys;

    // instance constructor
    public Person(string name, Professor supervisor) : this(name) { this.Enroll(supervisor); }
}

Person mads = new Person("Mads Torgersen", tonyHoare);

WriteLine(mads.Supervisor);
```

```
extension Student extends Person
{
    // static field
    static Dictionary<Person, Professor> enrollees = new Dictionary<Person, Professor>();

    // instance method
    public void Enroll(Professor supervisor) { enrollees[this] = supervisor; }

    // instance property
    public Professor? Supervisor => enrollees.TryGetValue(this, out var supervisor) ? supervisor : null;

    // static property
    public static ICollection<Person> Students => enrollees.Keys;

    // instance constructor
    public Person(string name, Professor supervisor) : this(name) { this.Enroll(supervisor); }
}

Person mads = new Person("Mads Torgersen", tonyHoare);

WriteLine(mads.Supervisor);

var tonysStudents =
    from s in Person.Students
    where s.Supervisor == tonyHoare
    select s.Name;
```

```
extension Student extends Person
{
    // static field
    static Dictionary<Person, Professor> enrollees = new Dictionary<Person, Professor>();

    // instance method
    public void Enroll(Professor supervisor) { enrollees[this] = supervisor; }

    // instance property
    public Professor? Supervisor => enrollees.TryGetValue(this, out var supervisor) ? supervisor : null;

    // static property
    public static ICollection<Person> Students => enrollees.Keys;

    // instance constructor
    public Person(string name, Professor supervisor) : this(name) { this.Enroll(supervisor); }
}

class Professor { … }

interface IStudent
{
    string Name { get; }
    Professor? Supervisor { get; }
    void Enroll(Professor supervisor);
}
```

```
extension Student extends Person : IStudent
{
    // static field
    static Dictionary<Person, Professor> enrollees = new Dictionary<Person, Professor>();

    // instance method
    public void Enroll(Professor supervisor) { enrollees[this] = supervisor; }

    // instance property
    public Professor? Supervisor => enrollees.TryGetValue(this, out var supervisor) ? supervisor : null;

    // static property
    public static ICollection<Person> Students => enrollees.Keys;

    // instance constructor
    public Person(string name, Professor supervisor) : this(name) { this.Enroll(supervisor); }
}

class Professor { … }

interface IStudent
{
    string Name { get; }
    Professor? Supervisor { get; }
    void Enroll(Professor supervisor);
}
```

```
extension Student extends Person : IStudent
{
    // static field
    static Dictionary<Person, Professor> enrollees = new Dictionary<Person, Professor>();

    // instance method
    public void Enroll(Professor supervisor) { enrollees[this] = supervisor; }

    // instance property
    public Professor? Supervisor => enrollees.TryGetValue(this, out var supervisor) ? supervisor : null;

    // static property
    public static ICollection<Person> Students => enrollees.Keys;

    // instance constructor
    public Person(string name, Professor supervisor) : this(name) { this.Enroll(supervisor); }
}

class Professor { … }

interface IStudent
{
    string Name { get; }
    Professor? Supervisor { get; }
    void Enroll(Professor supervisor);
}
```

```
extension Student extends Person : IStudent
{
    // static field
    static Dictionary<Person, Professor> enrollees = new Dictionary<Person, Professor>();

    // instance method
    public void Enroll(Professor supervisor) { enrollees[this] = supervisor; }

    // instance property
    public Professor? Supervisor => enrollees.TryGetValue(this, out var supervisor) ? supervisor : null;

    // static property
    public static ICollection<Person> Students => enrollees.Keys;

    // instance constructor
    public Person(string name, Professor supervisor) : this(name) { this.Enroll(supervisor); }
}

class Professor { … }

interface IStudent
{
    string Name { get; }
    Professor? Supervisor { get; }
    void Enroll(Professor supervisor);
}
```

```
interface IGroup<T>
{
    static T Zero { get; }
    static T operator +(T t1, T t2);
}

extension IntGroup extends int : IGroup<int>
{
     public static int T Zero => 0;
}

public static T AddAll<T>(T[] ts) where T : IGroup<T>
{
    T result = T.Zero;
    foreach (T t in ts) { result += t; }
    return result;
}

int sixtyThree = AddAll(new [] { 1, 2, 4, 8, 16, 32 });
```

docs.microsoft.com/dotnet/csharp

blogs.msdn.microsoft.com/dotnet

github.com/dotnet/csharplang