WELCOME TO

PARTSTRADER

MARKETS LIMITED

Welcome to PartsTrader

- Where is the food?
- Where are the restrooms?
- What does PartsTrader do?
- Who is talking to us?
- Who else is here?

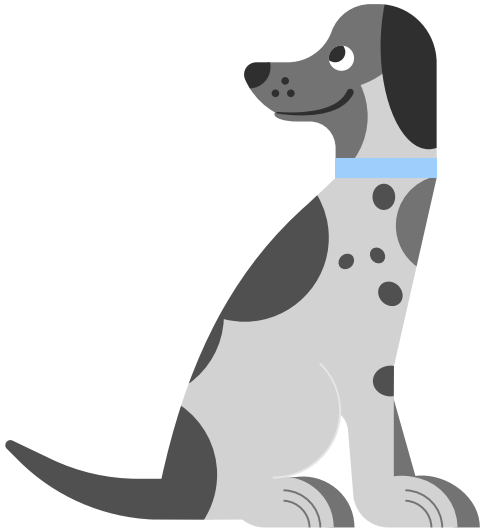
PARTSTRADER
MARKETS LIMITED

NULLWORRIES

(BE HAPPY)

Overview

- Why do we use null?
- How does it cause problems?
- What can we do about it?

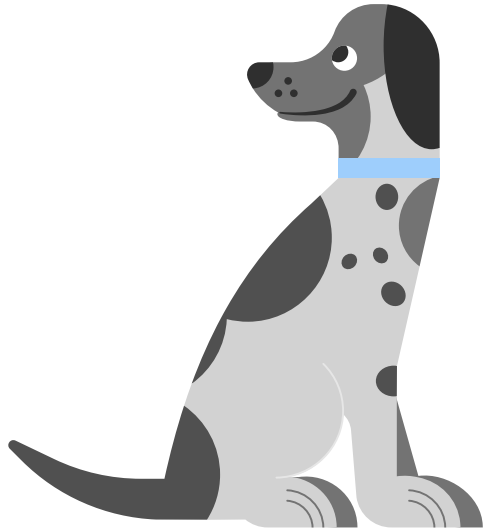


PARTSTRADER
MARKETS LIMITED

Why do we use null?

We use null to represent:

- the *absence* of a value, or
- an *unspecified* value.



PARTSTRADER
MARKETS LIMITED

Absence of a value

The *absence* of a value represents:

- an *unknown* value, or
- an *unnecessary* value.

```
public class Person
{
    public Person(string givenName, string familyName)
    …
}

// Jane Citizen
Person a = new (“Jane”, “Citizen”);

// John ?
Person b = new(“John”, null);

// Lorde
Person c = new(“Lorde”, null);
```

PARTSTRADER
MARKETS LIMITED

Unspecified value

An *unspecified value* represents:

- an *invalid* value, or
- a *choice* to omit a value.

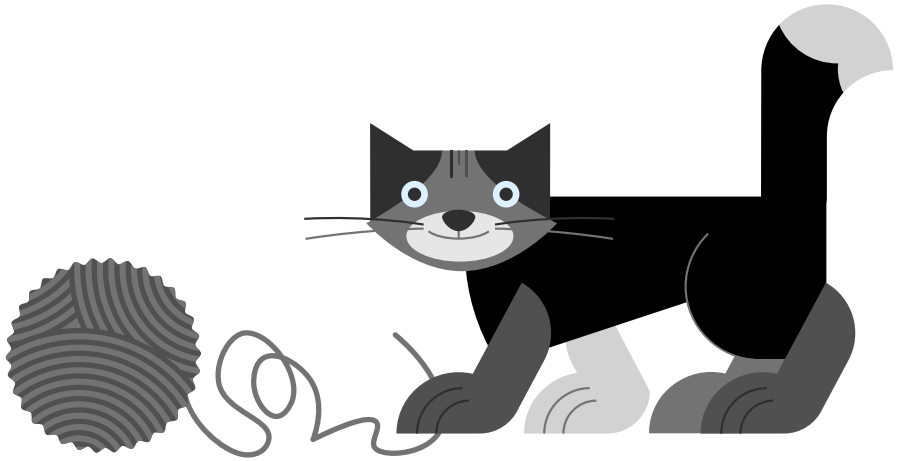
```
public class Person
{
    public Person(DateTime? dateOfBirth)
    {
        DateOfBirth = dateOfBirth != null &&
            dateOfBirth.Value < DateTime.Now
                ? dateOfBirth : null;
    }

    public DateTime? DateOfBirth { get; }
}

Person a = new(DateTime.Parse(“3000-04-01”));

Person b = new(null);
```

PARTSTRADER
MARKETS LIMITED

How does it cause problems?

Null causes problems due to:

- uncertainty, and
- null references.



PARTSTRADER
MARKETS LIMITED

Uncertainty

It is not clear if *null* represents:

- an *unknown* value,
- an *unnecessary* value,
- an *invalid* value, or
- a *choice* to omit a value.

```
public class DateRange
{
    public DateRange(DateTime? start, DateTime? end)
    {
        Start = start
        End = end
    }

    public DateTime? Start { get; }
    public DateTime? End { get; }
}

// what does this mean?
DateRange range = new(null, null);
```

PARTSTRADER
MARKETS LIMITED

What is a null reference?

A reference points to *something* in memory.

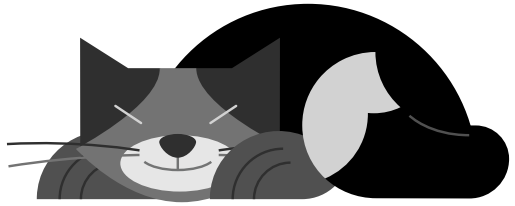
```
// reference to object
object obj = new();

// reference to string
var str = “Hello world!”;
```

PARTSTRADER
MARKETS LIMITED

What is a null reference?

A reference points to *something* in memory.

A null reference points at *nothing*.



```
// reference to object
object obj = new();

// reference to string
var str = “Hello world!”;

// null reference
object obj = null;
```

PARTSTRADER
MARKETS LIMITED

Is that a problem?

Tony Hoare invented Quicksort in 1959.

In 1965, he invented null references.

He apologised for them in 2009…

PARTSTRADER
MARKETS LIMITED

Why did he do that?

"I call it my billion dollar mistake"

"I couldn't resist the temptation to put in a null reference, because it was so easy to implement"

"This has led to innumerable errors, vulnerabilities and system crashes"

PARTSTRADER
MARKETS LIMITED

What is the problem?

~~Developers are lazy~~

Developers are human, and make mistakes

It is *very easy* to forget to check for null references

```
var obj = GetObject();
var text = obj.ToString();

var n = text.Length;

Regex pattern = new (@"^\\d{4,}$");
var isMatch = pattern.IsMatch(text);
```

alistair-grant/summer-of-tech/
null-worries/src/easy-to-forget

PARTSTRADER
MARKETS LIMITED

What happens if we forget?

For .NET, invoking a member on a null reference:

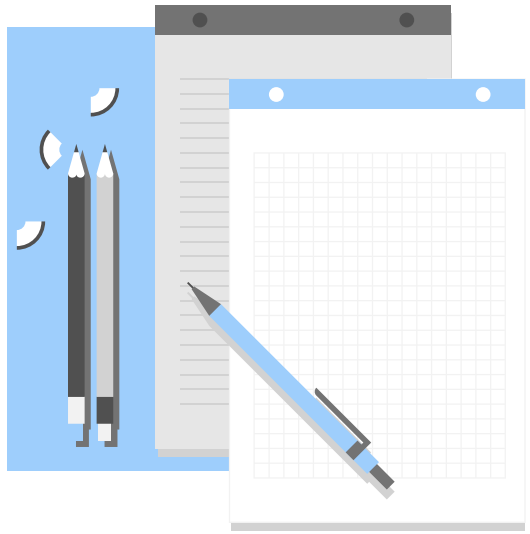
- will cause a *NullReferenceException*,
- will abort the current operation,
- may cause data/state corruption, and
- may terminate the application.

PARTSTRADER
MARKETS LIMITED

What can we do about it?

Null worries? Be happy with:

- alternatives to null,
- ensuring valid object state,
- C# language features, and
- functional programming.



PARTSTRADER
MARKETS LIMITED

Default objects

If object not found, then return:

- null.

```
public class Person
{
    public static Person GetPerson(int id)
    {
        // not found
        return null;
    }
}

var person = Person.GetPerson(1);

// exception: person is null
var text = person.ToString();
```

alistair-grant/summer-of-tech/
null-worries/src/default-objects

PARTSTRADER
MARKETS LIMITED

Default objects

If object not found, then return:

- ~~null.~~
- a default object.

```
public class Person
{
    public static readonly Person Unknown = new();

    public static Person GetPerson(int id)
    {
        // not found
        return Unknown;
    }
}

var person = Person.GetPerson(1);

// no exception
var text = person.ToString();
```

alistair-grant/summer-of-tech/
null-worries/src/default-objects

PARTSTRADER
MARKETS LIMITED

Are default objects practical?

Advantages:

- null worry eliminated.

Disadvantages:

- non-standard code, and
- easy to forget about.

PARTSTRADER
MARKETS LIMITED

Enforce valid object state

If an object is initialised with an invalid state, then:

- keep calm and carry on.

```
public class Book
{
    public Book(string title)
    {
        Title = title;
    }

    public string Title { get; }
}

// no exception
Book book = new(null);

// exception: Title is null
var searchTerm = “the”;
var isMatch = book.Title.Contains(searchTerm);
```

alistair-grant/summer-of-tech/
null-worries/src/valid-state

PARTSTRADER
MARKETS LIMITED

Enforce valid object state

If an object is initialised with an invalid state, then:

- ~~keep calm and carry on.~~
- fail fast.

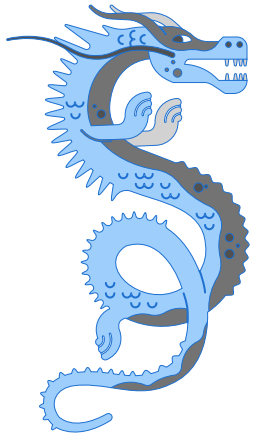
```
public class Book
{
    public Book(string title)
    {
        Title = title ?? throw new
            ArgumentNullException(nameof(title));
    }

    public string Title { get; }
}

// exception: fail fast
Book a = new(null);

// no exception
Book b = new("Screens");
```

alistair-grant/summer-of-tech/
null-worries/src/valid-state

PARTSTRADER
MARKETS LIMITED

Beware of value types

Value types in C# always have a default constructor!



```
public struct Book
{
    public Book(string title)
    {
        Title = title ?? throw new
            ArgumentNullException(nameof(title));
    }

    public string Title { get; }
}

// has a default constructor!
Book book = new();

// exception: Title is null
var searchTerm = "the";
var isMatch = book.Title.Contains(searchTerm);
```

alistair-grant/summer-of-tech/
null-worries/src/valid-state

PARTSTRADER
MARKETS LIMITED

Enforce valid object state

If a choice exists between memory and state, then:

- choose memory.

```
public class Author
{
    public Author()
    {
        // use default initializers
    }

    public Author(IEnumerable<Book> books)
    {
        Books = books ?? throw new
            ArgumentNullException(nameof(books));
    }

    public IEnumerable<Book> Books { get; };
}

Author author = new();

// exception: Books is null
var hasBooks = author.Books.Any();
```

alistair-grant/summer-of-tech/
null-worries/src/valid-state

PARTSTRADER
MARKETS LIMITED

Enforce valid object state

If a choice exists between memory and state, then:

- ~~choose memory.~~
- choose valid object state.

```
public class Author
{
    public Author()
    {
        Books = Enumerable.Empty<Book>();
    }

    public Author(IEnumerable<Book> books)
    {
        Books = books ?? throw new
            ArgumentNullException(nameof(books));
    }

    public IEnumerable<Book> Books { get; };
}

Author author = new();

// no exception
var hasBooks = author.Books.Any();
```

alistair-grant/summer-of-tech/
null-worries/src/valid-state

PARTSTRADER
MARKETS LIMITED

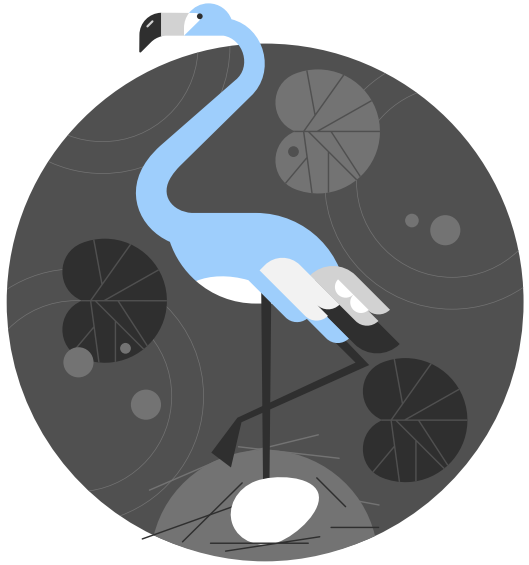
Is enforcing state practical?

Disadvantages:

- failures will happen.

Advantages:

- known failure points, and
- principal of least surprise.



PARTSTRADER
MARKETS LIMITED

Use setters to enforce state

If a property is set to an invalid state, then:

- keep calm and carry on.

```
public class Book
{
    public Book(string title)
    {
        Title = title;
    }

    public string Title { get; set; }
}

// no exception
Book book = new("Screens");
book.Title = null;

// exception: Title is null
var searchTerm = "the";
var isMatch = book.Title.Contains(searchTerm);
```

PARTSTRADER
MARKETS LIMITED

Use setters to enforce state

If a property is set to an invalid state, then:

- ~~keep calm and carry on.~~
- fail fast.

```
public class Book
{
    public Book(string title)
    {
        Title = title;
    }

    private string _title;
    public string Title
    {
        get => _title;
        set => _title = value ?? throw new
            ArgumentNullException(nameof(title));
    }
}

Book book = new("Screens");

// exception: fail fast
Book.Title = null;
```

PARTSTRADER MARKETS LIMITED

Is enforcing state with setters practical?

Advantages:

- known failures points.

Disadvantages:

- additional failure points, and
- greater chance of surprises.

PARTSTRADER
MARKETS LIMITED

Mutable and immutable types

The value of a *mutable* type *can* be changed

```
public class Author
{
    public Author(string name)
    {
        Name = name;
    }

    public string Name { get; set; }
}

Author author = new("Christopher Laine");

// value can be changed
author.Name = "Peter F. Hamilton";
```

PARTSTRADER
MARKETS LIMITED

Mutable and immutable types

The value of a *mutable* type *can* be changed.

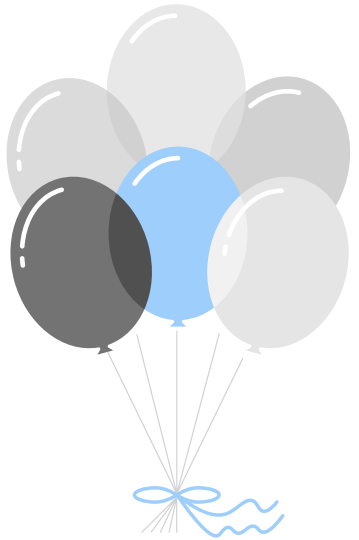
The value of an *immutable* type *cannot* be changed.

```
public class Author
{
    public Author(string name)
    {
        Name = name;
    }

    public string Name { get; }
}

Author author = new("Christopher Laine");

// compiler error: value cannot be changed
author.Name = "Peter F. Hamilton";
```

PARTSTRADER
MARKETS LIMITED

Advantages of immutable types

- Enforce valid object state with constructors
- Safer multi-threading:
  - ✓ Cannot be changed by other threads
  - ✓ Does not require synchronisation
- WYSIWYG: principle of least surprise



PARTSTRADER
MARKETS LIMITED

Disadvantages of immutable types

- ~~Greater memory usage~~
- Can be painful to initialise

```
public class Book
{
    public Book(
        string title,
        string isbn,
        string subject,
        decimal retailPrice,
        Author author,
        IEnumerable<Review> reviews,
        …)
    {
        // TODO: validate every argument...
    }
}
```

PARTSTRADER
MARKETS LIMITED

Playing together: the builder pattern

The builder pattern:

- allows bit-by-bit initialisation,
- guides us through it, and
- can build an immutable type.

PARTSTRADER
MARKETS LIMITED

Start with a part

- Immutable type
- Not enforcing valid state
- Internal constructor
- Not a value type

```
public class Part
{
    internal Part(
        string partNumber,
        string description,
        decimal listPrice)
    {
        PartNumber = partNumber;
        Description = description;
        ListPrice = listPrice;
    }

    public string PartNumber { get; }

    public string Description { get; }

    public decimal ListPrice { get; }
}
```

alistair-grant/summer-of-tech/
null-worries/src/builders

PARTSTRADER
MARKETS LIMITED

Scaffold the part builder

Required:

- part number, and
- list price.

Optional:

- description.

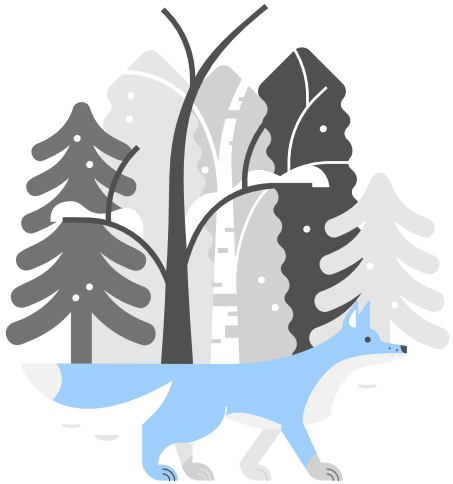
```
public class PartBuilder
{
    private string _partNumber;
    private string _description;
    private decimal _listPrice;

    public PartBuilder()
    {
        _description = string.Empty;
    }
}
```

alistair-grant/summer-of-tech/
null-worries/src/builders

PARTSTRADER
MARKETS LIMITED

Begin at the end

- Explicit interface implementation



```
public interface IListPriceInitialized
{
    Part Build();
}

public class PartBuilder : IListPriceInitialized
{
    Part IListPriceInitialized.Build() =>
        new(_partNumber, _description, _listPrice);
}
```

alistair-grant/summer-of-tech/
null-worries/src/builders

PARTSTRADER
MARKETS LIMITED

A required list price

- Code omitted for clarity
- Enforcing valid state
- Returns available actions

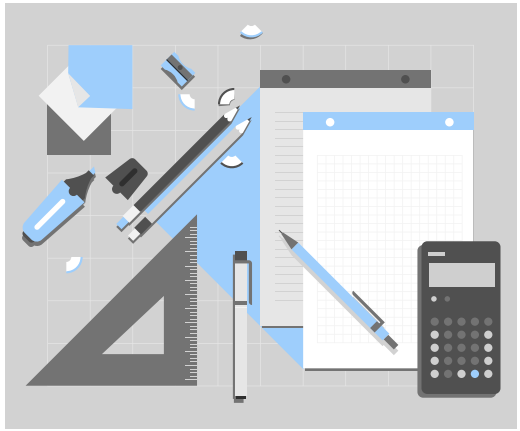
```
public interface IDescriptionInitialized
{
    IListPriceInitialized ListPrice(decimal listPrice);
}

public class PartBuilder : IDescriptionInitialized, …
{
    IListPriceInitialized
        IDescriptionInitialized.ListPrice(
            decimal listPrice)
    {
        if (listPrice < 0m)
        {
            throw new ArgumentOutOfRangeException(…);
        }

        _listPrice = listPrice;

        return this;
    }
}
```

alistair-grant/summer-of-tech/
null-worries/src/builders

PARTSTRADER
MARKETS LIMITED

An optional description

What happens if *null* is passed for the description?

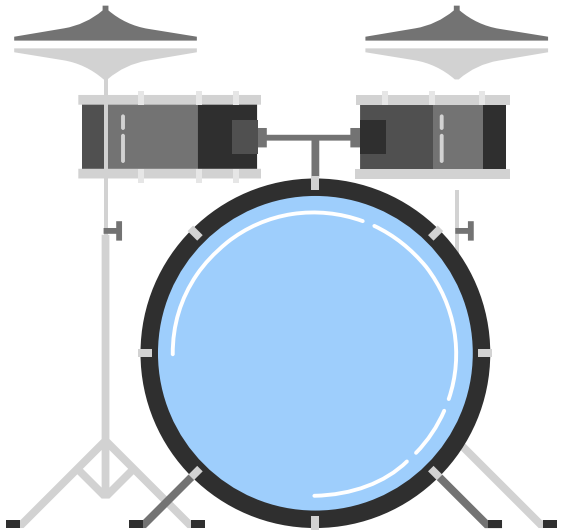


```
public interface IPartNumberInitialized :
    IDescriptionInitialized
{
    IDescriptionInitialized Description(
        string description);
}

public class PartBuilder : IPartNumberInitialized, …
{
    IDescriptionInitialized
        IPartNumberInitialized.Description(
            string description)
    {
        if (string.IsNullOrWhiteSpace(description))
        {
            throw new ArgumentException(…);
        }

        _description = description;

        return this;
    }
}
```

alistair-grant/summer-of-tech/
null-worries/src/builders

PARTSTRADER
MARKETS LIMITED

A required part number



```
public interface IUninitialized
{
    IPartNumberInitialized PartNumber(
        string partNumber);
}

public class PartBuilder : IUninitialized, …
{
    public IPartNumberInitialized PartNumber(
        string partNumber)
    {
        if (string.IsNullOrWhiteSpace(partNumber))
        {
            throw new ArgumentException(…);
        }

        _partNumber = partNumber;

        return this;
    }
}
```

alistair-grant/summer-of-tech/
null-worries/src/builders

PARTSTRADER
MARKETS LIMITED

What have we got?

- ✓ Bit-by-bit initialisation
- ✓ Guides us through it
- ✓ Creates immutable type
- ✓ Enforces valid object state

```
// with description
var a = new PartBuilder()
    .PartNumber("1234-abcd")
    .Description("Sample Part")
    .ListPrice(12.95)
    .Build();

// no description
var b = new PartBuilder()
    .PartNumber("5678-efgh")
    .ListPrice(3.14)
    .Build();

// compiler error
var c = new PartBuilder()
    .PartNumber("1234-abcd")
    .Build();
```

alistair-grant/summer-of-tech/
null-worries/src/builders

PARTSTRADER
MARKETS LIMITED

C# language features

- Nullable reference types (C# 8)
- Records (C# 9)
- Init-only setters (C# 9)
- Null parameter checking (C# 10)*
- Required properties (C# 10) *

PARTSTRADER
MARKETS LIMITED

Nullable reference types

- C# 8 (.NET Core 3.x)
- Same syntax as value type
- Used to indicate when a reference can be *null*

```
// value type
int a = 0;

// nullable value type
int? b = null;

// “non-nullable” reference type
object c = new object();

// nullable reference type
object? d = null;
```

PARTSTRADER
MARKETS LIMITED

Disabled by default

- Edit project file
- Add <Nullable> element
- Set to *enable*

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <Nullable>enable</Nullable>
  </PropertyGroup>
</Project>
```

alistair-grant/summer-of-tech/
null-worries/src/nullable-references

PARTSTRADER
MARKETS LIMITED

How does it help?

- Shows a warning if a reference can be null
- Reveals unexpected edge cases in old code
- Suggestion: treat warnings as errors

```
// valid: can be null
object? a = null;

// invalid: cannot be null
object? b = null;

object obj = new();
var text = obj.ToString();

// CS8602 Dereference of a possibly null reference
var n = text.Length;

if (text != null)
{
    // no warning: text is non-null
    var m = text.Length;
}
```

alistair-grant/summer-of-tech/
null-worries/src/nullable-references

PARTSTRADER
MARKETS LIMITED

Records

- C# 9 (.NET 5.x)
- Automatic properties
- Does *not* automatically enforce valid object state
- Does automatically give Equals, ToString, etc…

```
// class
public class Person
{
    public Person(string givenName, string familyName)
    {
        GivenName = givenName;
        FamilyName = familyName;
    }

    public string GivenName { get; }

    public string FamilyName { get; }
}

// record
public record Person
{
    public Person(string GivenName, string FamilyName);
}
```

PARTSTRADER MARKETS LIMITED

Init-only setters

- C# 9 (.NET 5.x)
- Immutable properties
- Optional initialization only
- Use a backing field to enforce property state

```
public class Part
{
    public string PartNumber { get; init; }

    public string Description { get; init; }

    public decimal ListPrice { get; init; }
}

Part part = new
{
    PartNumber = "1234-abcd",
    ListPrice = 12.95
};

// compiler error
part.Description = "Sample Part";
```

PARTSTRADER
MARKETS LIMITED

Null parameter checking

- C# 10 (.NET 6.x)*
- Prototype feature/syntax
- Only checks for *null*

```
// manual
public Book(string title)
{
    if (title == null)
    {
        throw new ArgumentNullException(…);
    }
    …
}

// automatic
public Book(string title!!)
{
    …
}
```

PARTSTRADER
MARKETS LIMITED

Required properties

- C# 10 (.NET 6.x)*
- Prototype feature/syntax
- Use backing fields to enforce valid object state

```
public class Part
{
    public required string PartNumber { get; init; }

    public string Description { get; init; }

    public required decimal ListPrice { get; init; }
}

// compiler error: ListPrice not initialized
Part part = new
{
    PartNumber = “1234-abcd”,
    Description = “Sample Part”
};
```

PARTSTRADER
MARKETS LIMITED

Functional programming

- Honest method signatures
- Option data type

PARTSTRADER
MARKETS LIMITED

Honest method signatures

- Failure throws exception
- Dishonest signature
- Principal of least surprise

```
// Int32 struct
public static int Parse(string s);

// success
var a = int.Parse("1");

// failure
var b = int.Parse("one");
```

PARTSTRADER
MARKETS LIMITED

Option data type

Also known as *Maybe* (Haskell)

Union of two types:

- *None* – value is absent
- *Some* – value is present

Enforces checking of None/Some

PARTSTRADER
MARKETS LIMITED

Building the option type

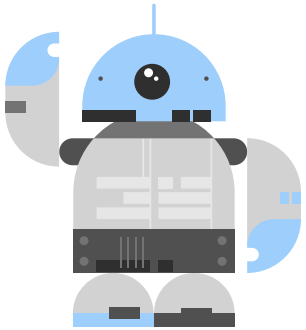
- Using value type
- No external access to underlying value

```
public struct Option<T>
{
    private readonly bool _hasValue;
    private readonly T _value;

    public TResult Match<TResult>(
        Func<TResult> none,
        Func<T, TResult> some)
    {
        return _hasValue ? some(_value) : none();
    }
}
```

alistair-grant/summer-of-tech/
null-worries/src/options

PARTSTRADER
MARKETS LIMITED

Building the none and some types



- Using value types
- No external access to underlying value

```
public struct None
{
    internal static readonly None Value = new();
}

public struct Some<T>
{
    internal Some(T value)
    {
        Value = value ?? throw new
            ArgumentNullException(nameof(value));
    }

    internal T Value { get; }
}
```

alistair-grant/summer-of-tech/
null-worries/src/options

PARTSTRADER
MARKETS LIMITED

Making it easy to use

- Code omitted for clarity
- Helper methods for None/Some syntax
- Implicit conversion of None, Some, value and reference types

```
public static class F
{
    public static None None => None.Value;

    public static Some<T> Some<T>(T value) =>
        new(value);
}

public struct Option<T>
{
    public static implicit operator
        Option<T>(None _) =>
            new();

    public static implicit operator
        Option<T>(Some<T> some) =>
            new(some.Value);

    public static implicit operator
        Option<T>(T value) =>
            value == null ? None : Some(value);
}
```

alistair-grant/summer-of-tech/
null-worries/src/options

PARTSTRADER
MARKETS LIMITED

A quick detour for a better view

```
public static class DictionaryExtensions
{
    public static Option<TValue>
        GetValue<TKey, TValue>(
            this IDictionary<TKey, TValue> source,
            TKey key) =>
                source.TryGetValue(key, out var value)
                    ? Some(value) : None;
}
```

alistair-grant/summer-of-tech/
null-worries/src/options

PARTSTRADER
MARKETS LIMITED

What have we got?

- ✓ Null worries?
- ✓ Explicit failure handling
- ✓ Success case handled
- ✓ Be happy!

```
var key = Guid.NewGuid().ToString();

Dictionary<string, string> dictionary =
    new() { { key, "Hello world!" } };

Console.WriteLine(
    dictionary.GetValue(key).Match(
        none: () => "Key not found",
        some: value => $"Value is \"{value}\"."));
```

alistair-grant/summer-of-tech/
null-worries/src/builders

PARTSTRADER
MARKETS LIMITED

Summary

- We use null when we do not have a better value to use
- It causes problems because others do not know what we mean by it
- We can do something about it by making our intentions clearer

PARTSTRADER
MARKETS LIMITED

How do we make our intentions clearer?

- Default objects
- Ensuring valid object state
- Immutable objects and builders
- C# language features
- Functional programming

THANKYOU

QUESTIONS?

alistair-grant/summer-of-tech/null-worries

PARTSTRADER
MARKETS LIMITED

alistair-grant/summer-of-tech/null-worries