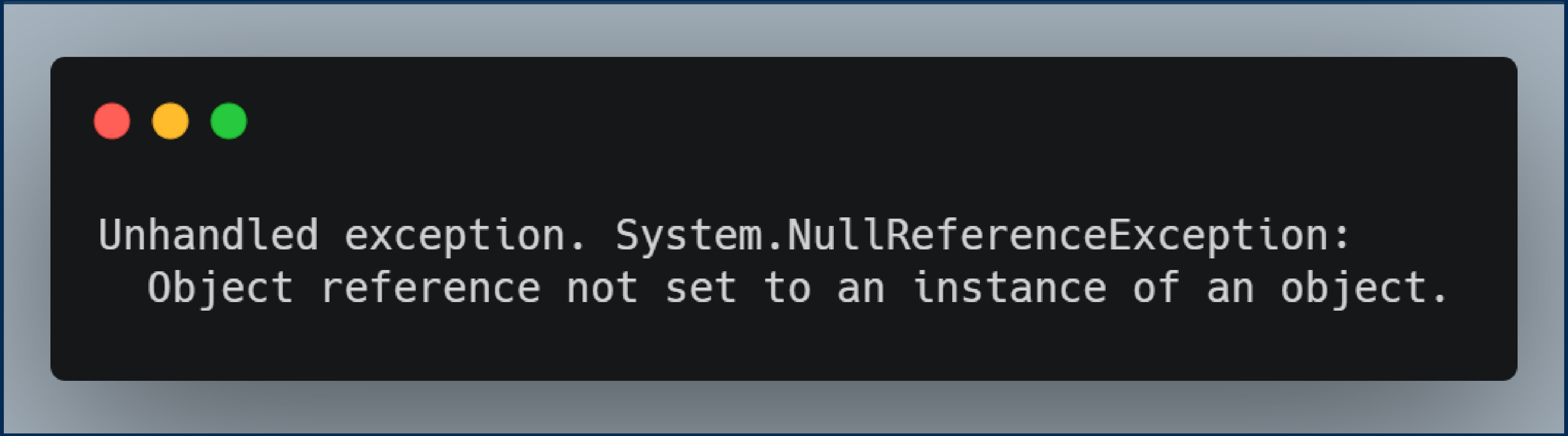
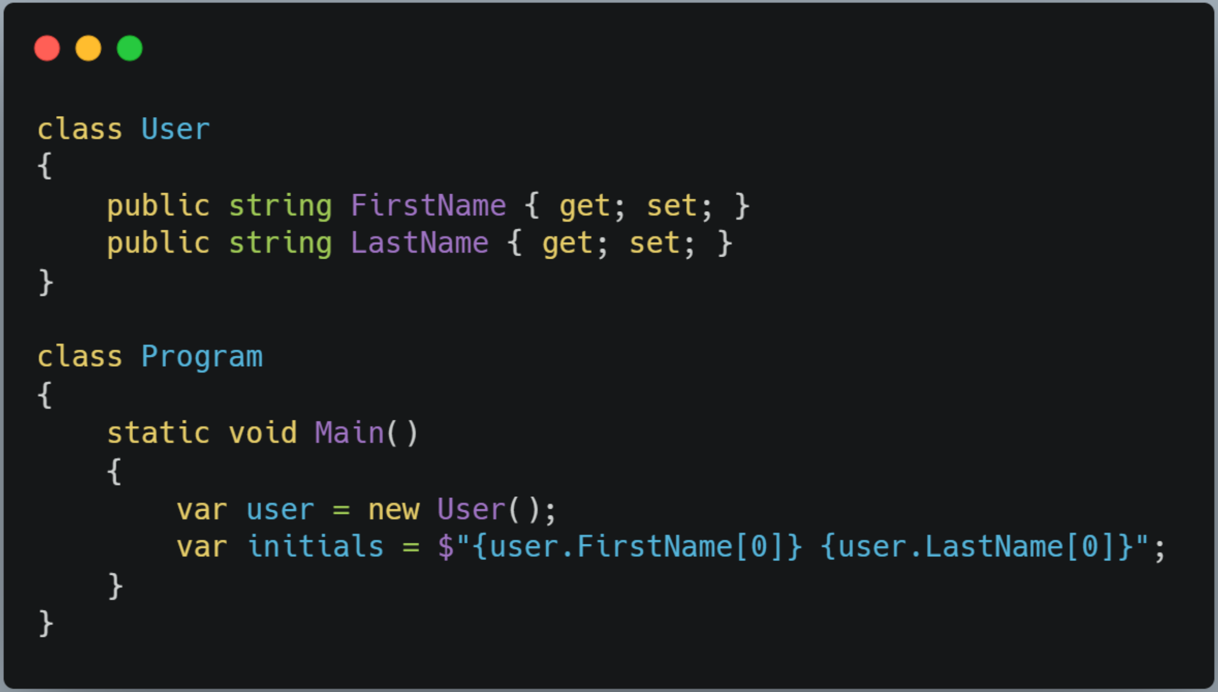
C# 8.0 AND NULLABLE REFERENCE TYPES

@kristofferjalen

tretton37

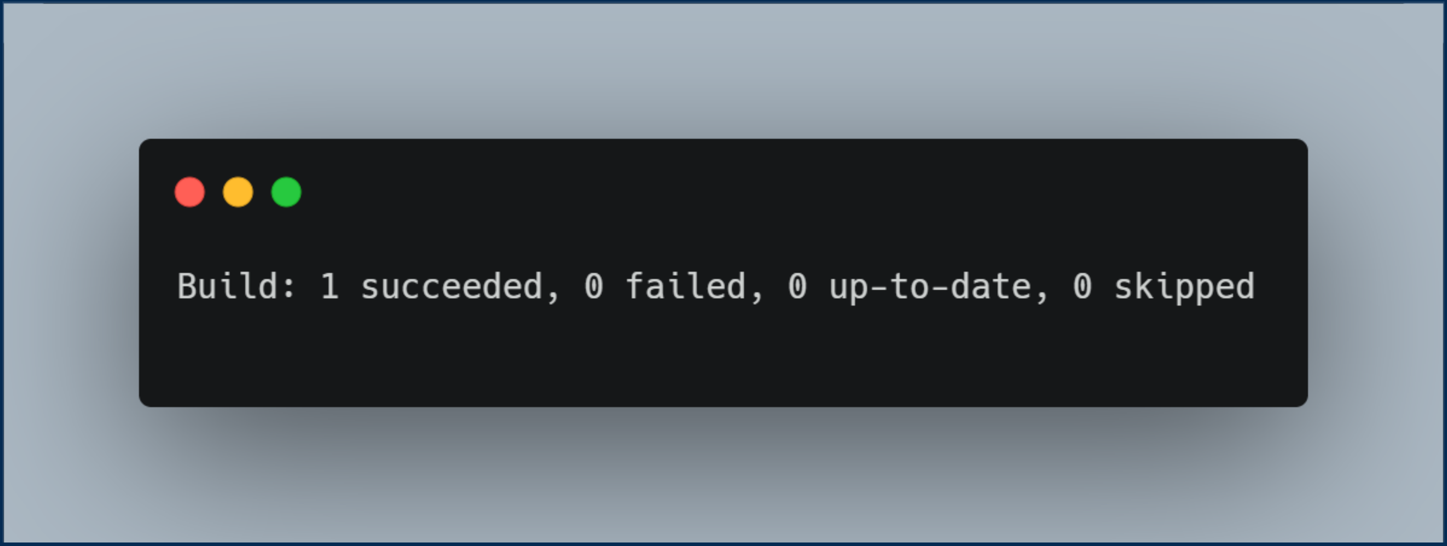


```
Unhandled exception. System.NullReferenceException:
  Object reference not set to an instance of an object.
```



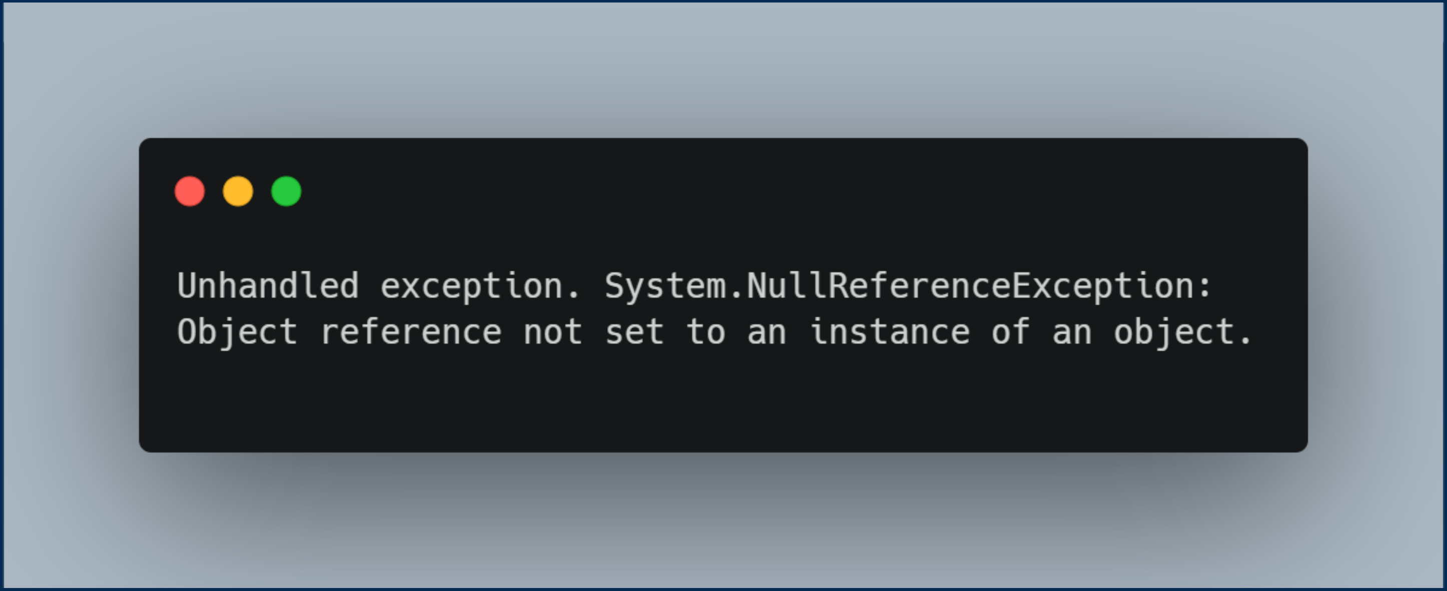
```
class User
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
}

class Program
{
    static void Main()
    {
        var user = new User();
        var initials = $"{user.FirstName[0]} {user.LastName[0]}";
    }
}
```

COMPILE



```
Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped
```

RUN



```
Unhandled exception. System.NullReferenceException:
Object reference not set to an instance of an object.
```

Get rid of the `NullReferenceException`

- nullable reference types
- non-nullable reference types

VALUE TYPE

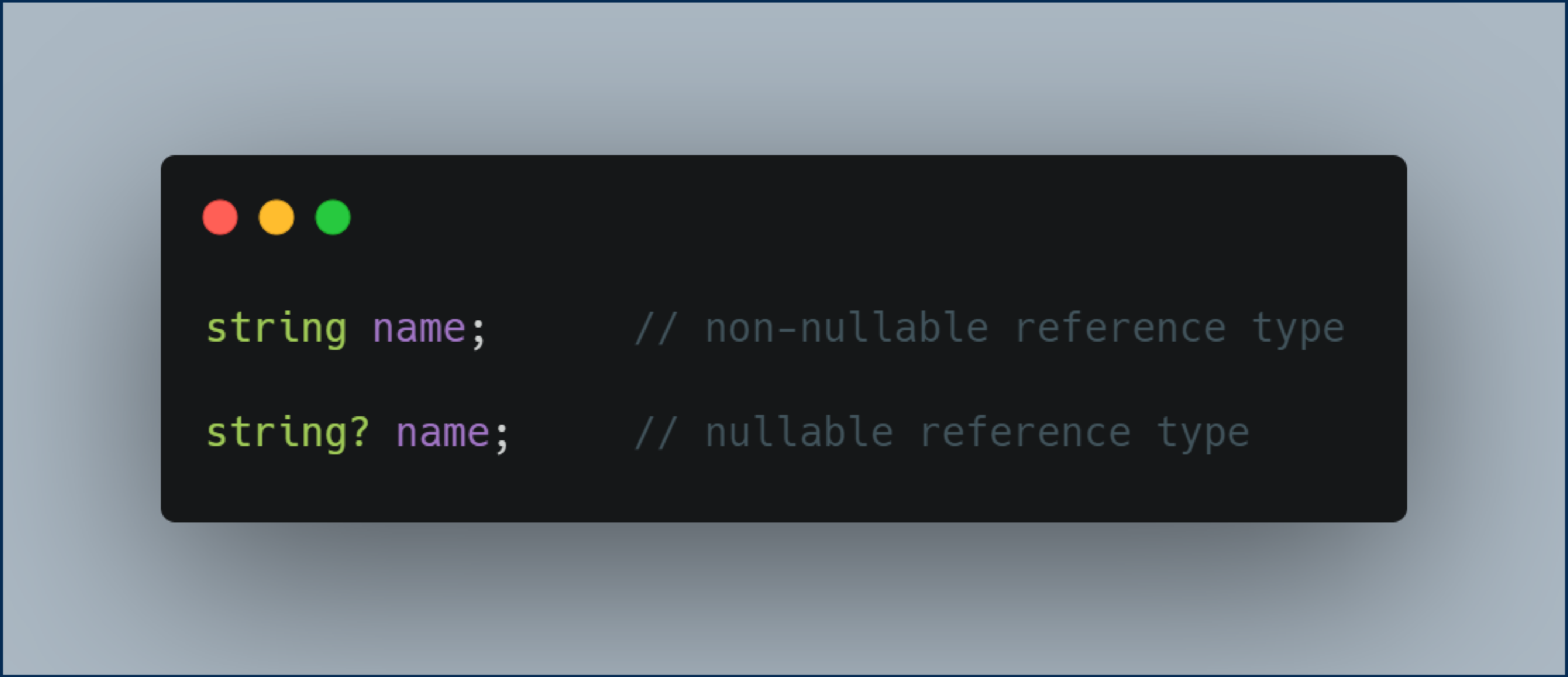
- A variable of a value type directly **contains** its **data**
- Assign a new value: **value** is copied
- Cannot be `null` by default
- `byte`, `int`, `float`, `double`, `decimal`, `bool`

REFERENCE TYPE

- Variable contains a **reference** to an object (instance of the type)
- Assign a new value: **reference** is copied, not the object itself
- Variables can reference the same object
- `class`, `interface`, `delegate`, `dynamic`, `object`, `string`

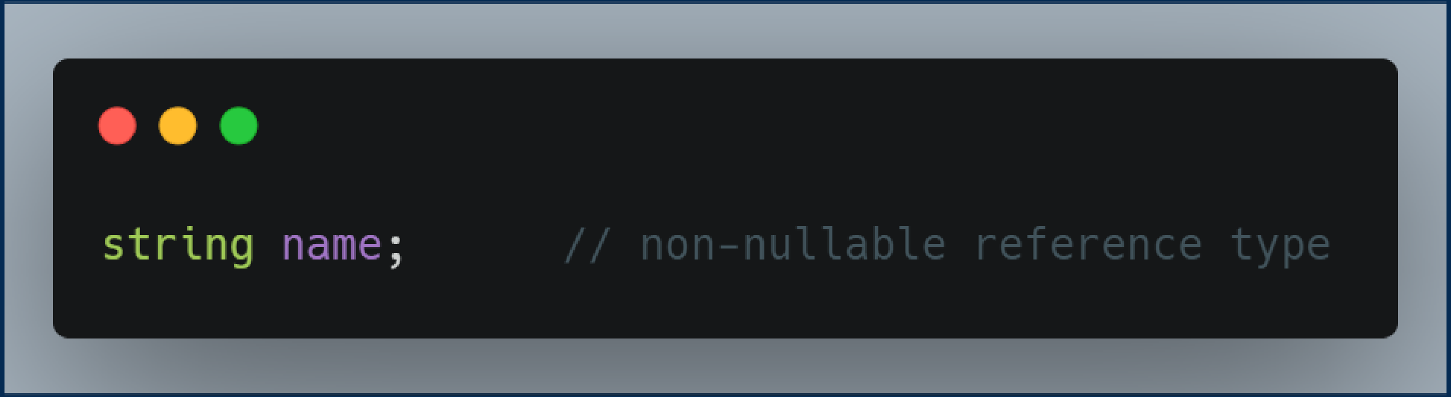
EXPRESS YOUR INTENT

- Some variables **must have** a value
- Some variables **may be missing** a value



```
string name;        // non-nullable reference type

string? name;       // nullable reference type
```

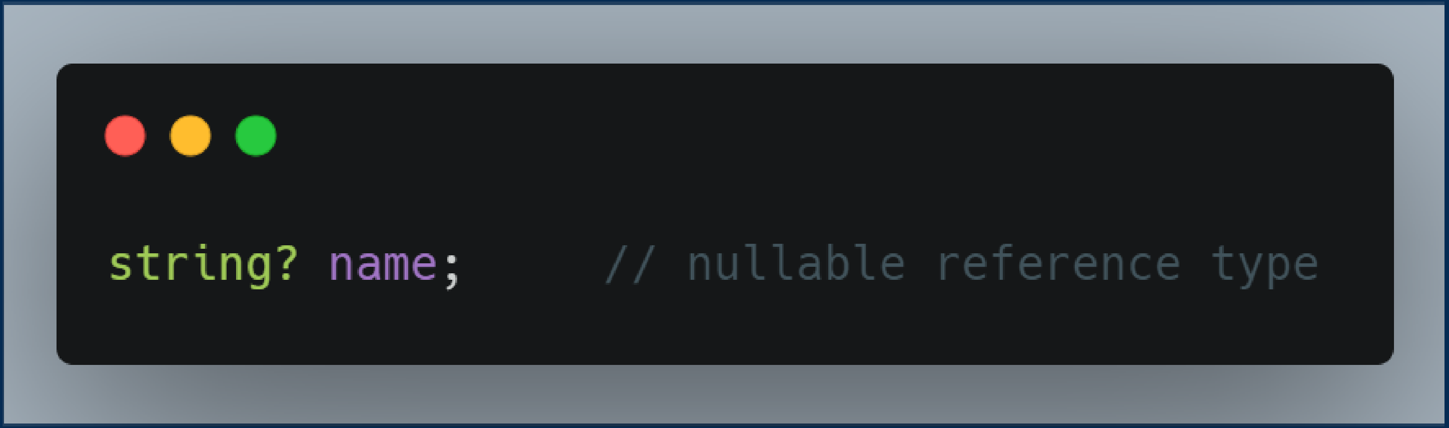
A reference is **not supposed** to be null



```
string name;       // non-nullable reference type
```

- Safe to **dereference**
- Must be initialized to a non-`null` value
- Can never be assigned the value `null`

A reference **may be null**



```
string? name;      // nullable reference type
```

- Compiler ensures you check for `null`
- May only be dereferenced when compiler can guarantee value isn't `null`
- May be assigned `null` value

BEFORE

No compiler warnings when:

- A reference type is initialized to `null`
- A reference type is assigned to `null`
- Reference types are dereferenced

NOW

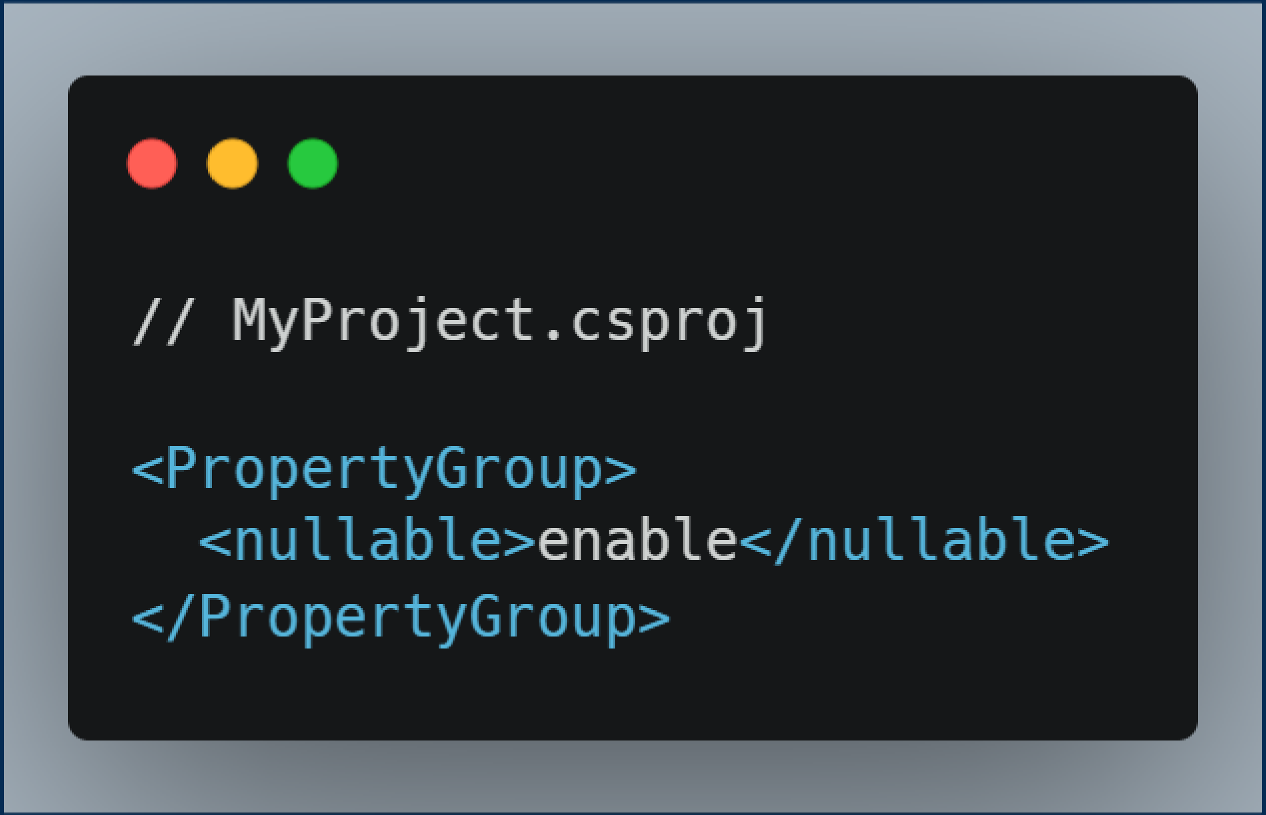
With **nullable references**:

- Suggestis marking variables as nullable
- Warnings when dereferencing a variable that may be `null`

ENABLE FEATURE

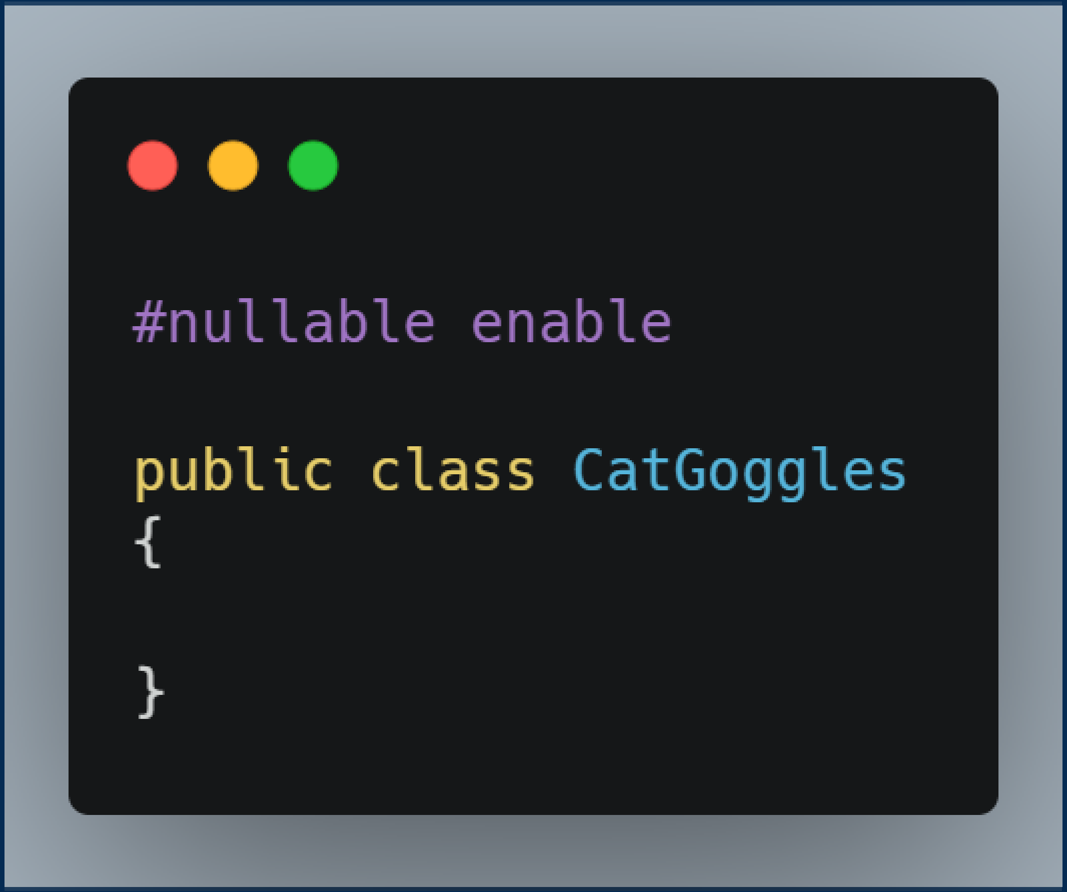
Breaking change in C#, hence opt-in.

TURN ON IN PROJECT



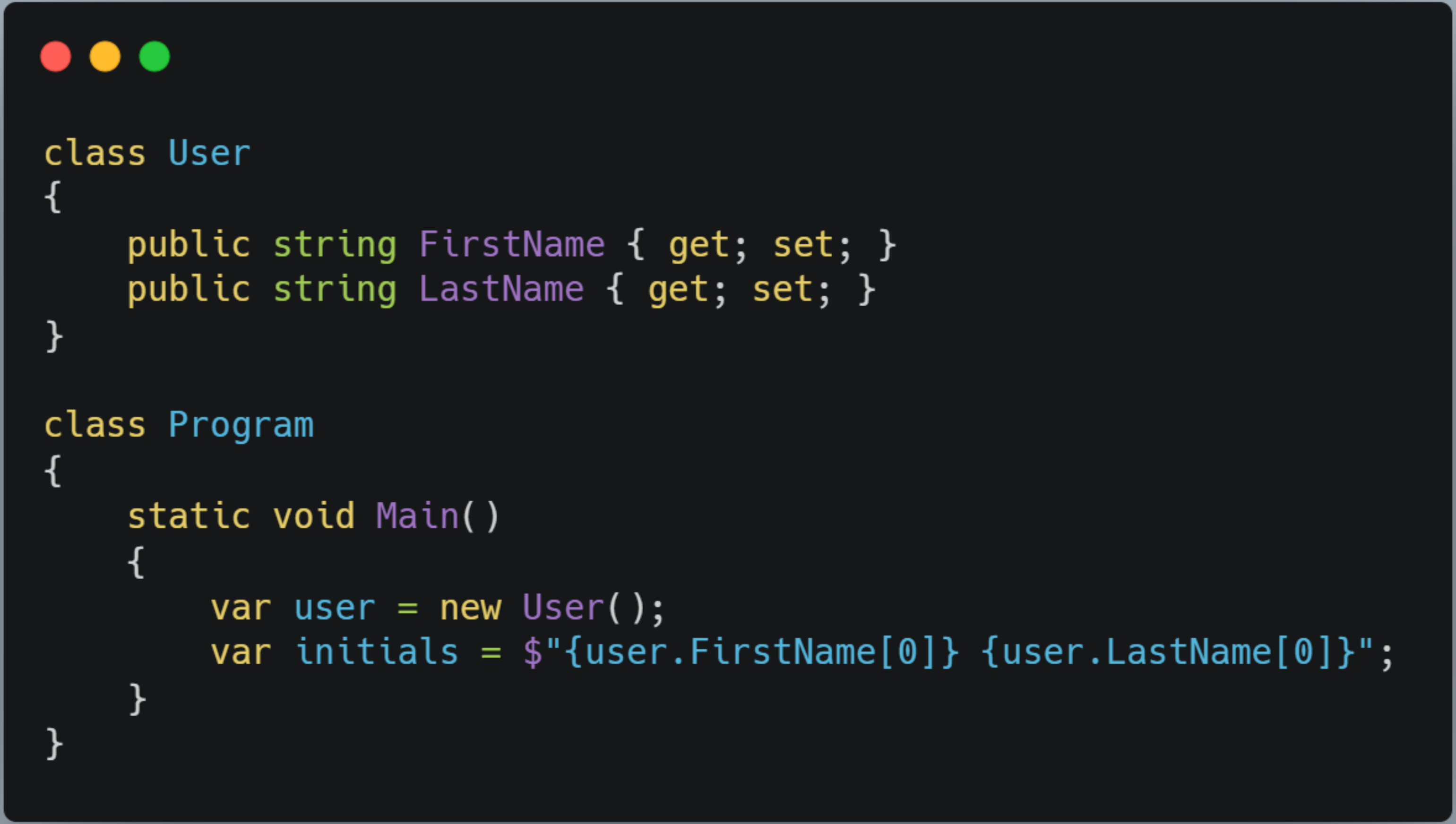
```
// MyProject.csproj

<PropertyGroup>
  <nullable>enable</nullable>
</PropertyGroup>
```

OR TEST IN SINGLE FILE



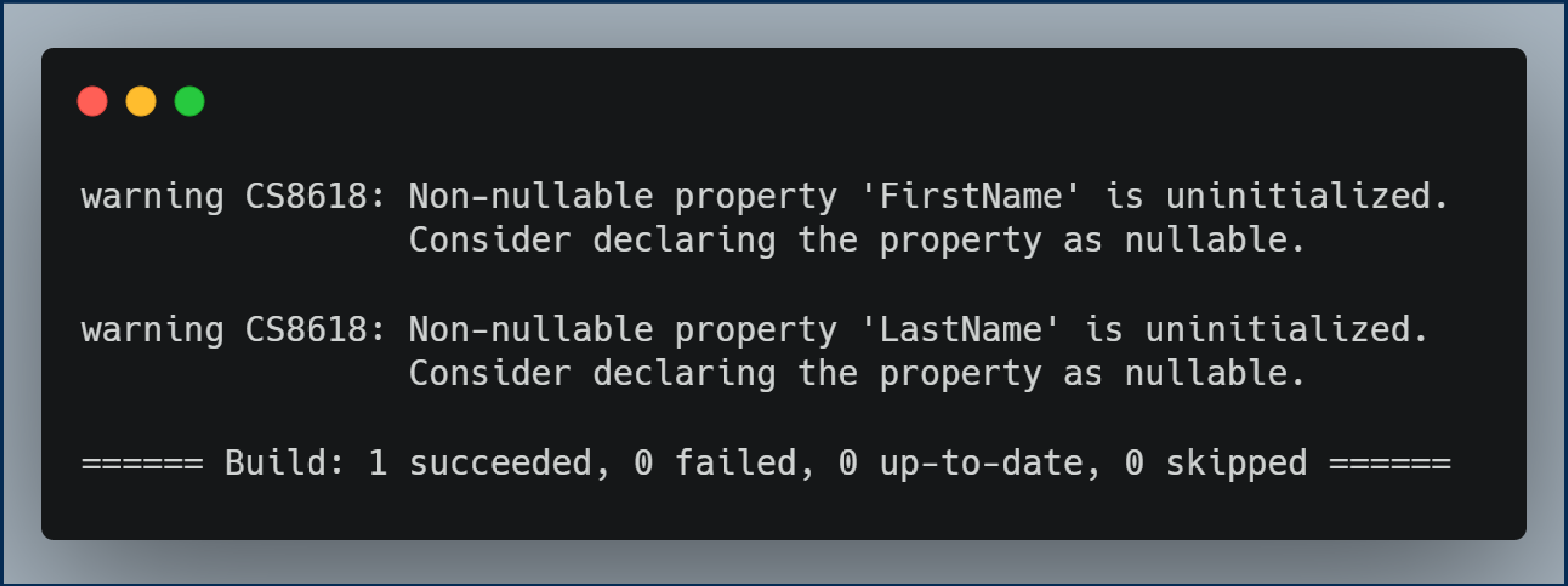
```
#nullable enable

public class CatGoggles
{

}
```

COMPILE



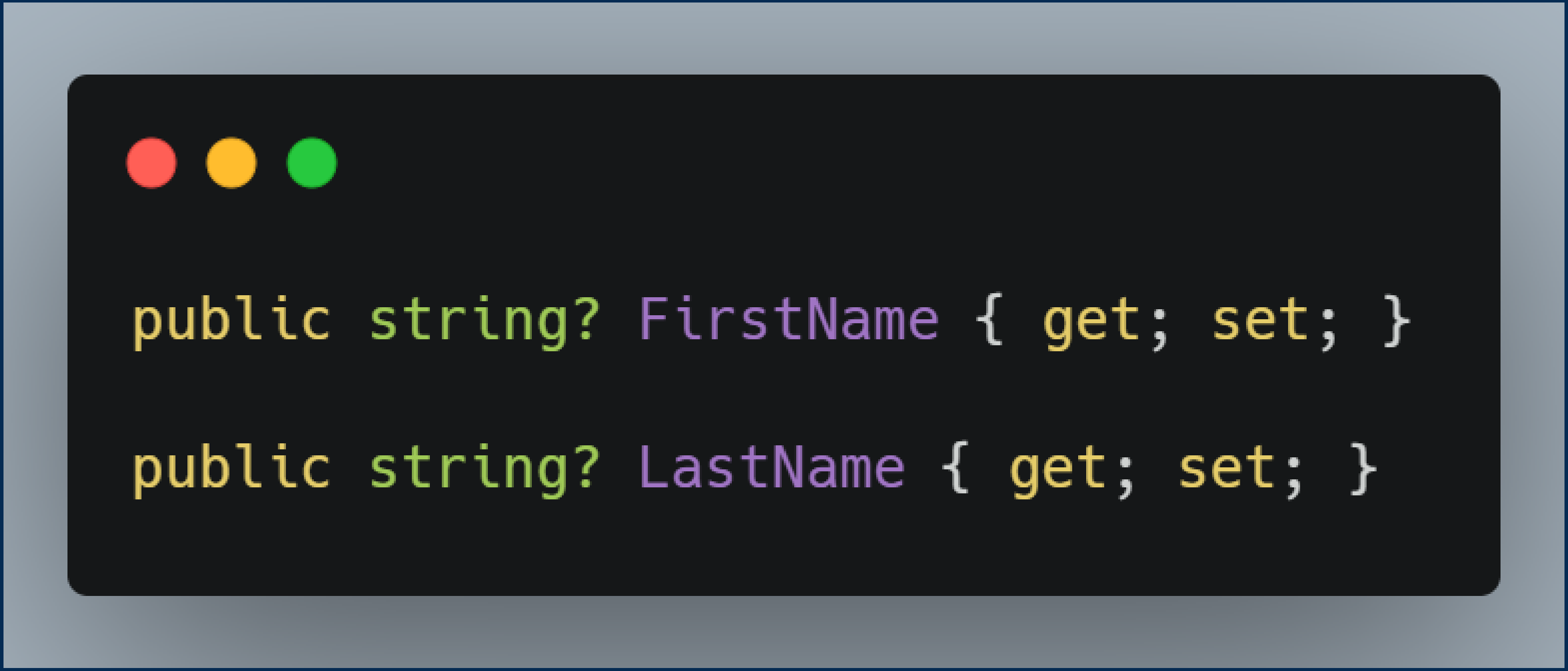
```
class User
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
}

class Program
{
    static void Main()
    {
        var user = new User();
        var initials = $"{user.FirstName[0]} {user.LastName[0]}";
    }
}
```

COMPILE



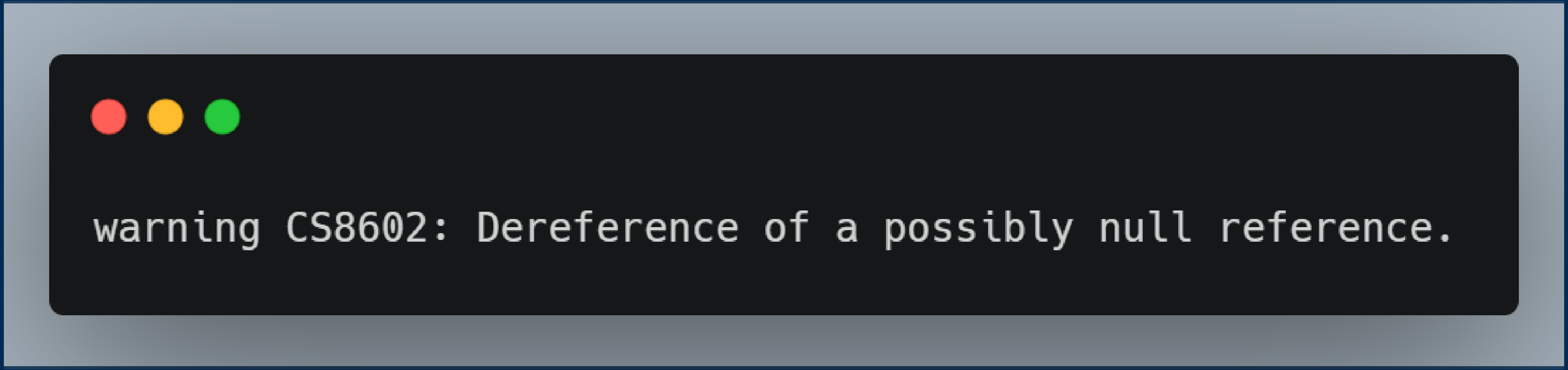
```
warning CS8618: Non-nullable property 'FirstName' is uninitialized.
                Consider declaring the property as nullable.

warning CS8618: Non-nullable property 'LastName' is uninitialized.
                Consider declaring the property as nullable.

====== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ======
```

DECLARE AS NULLABLE



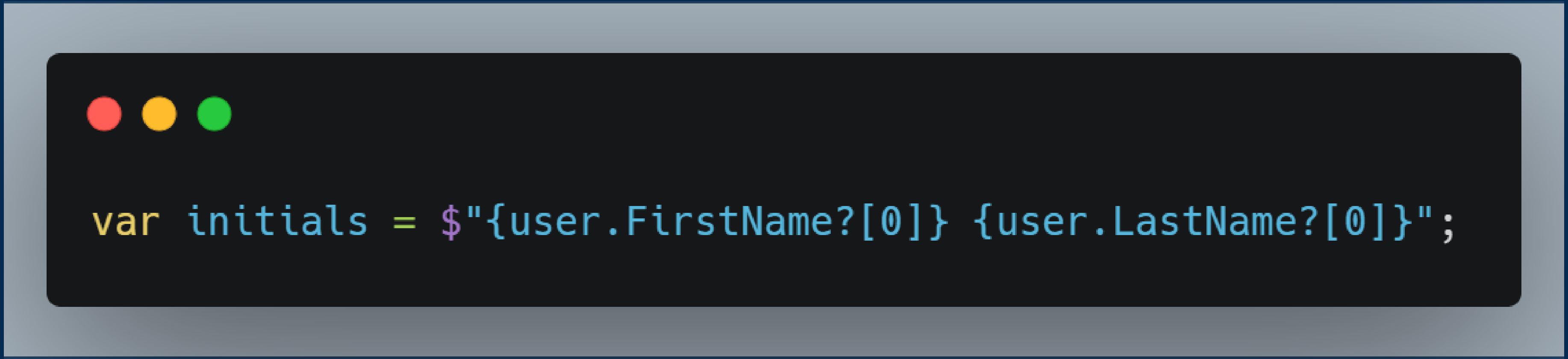
```
public string? FirstName { get; set; }

public string? LastName { get; set; }
```

BUILD



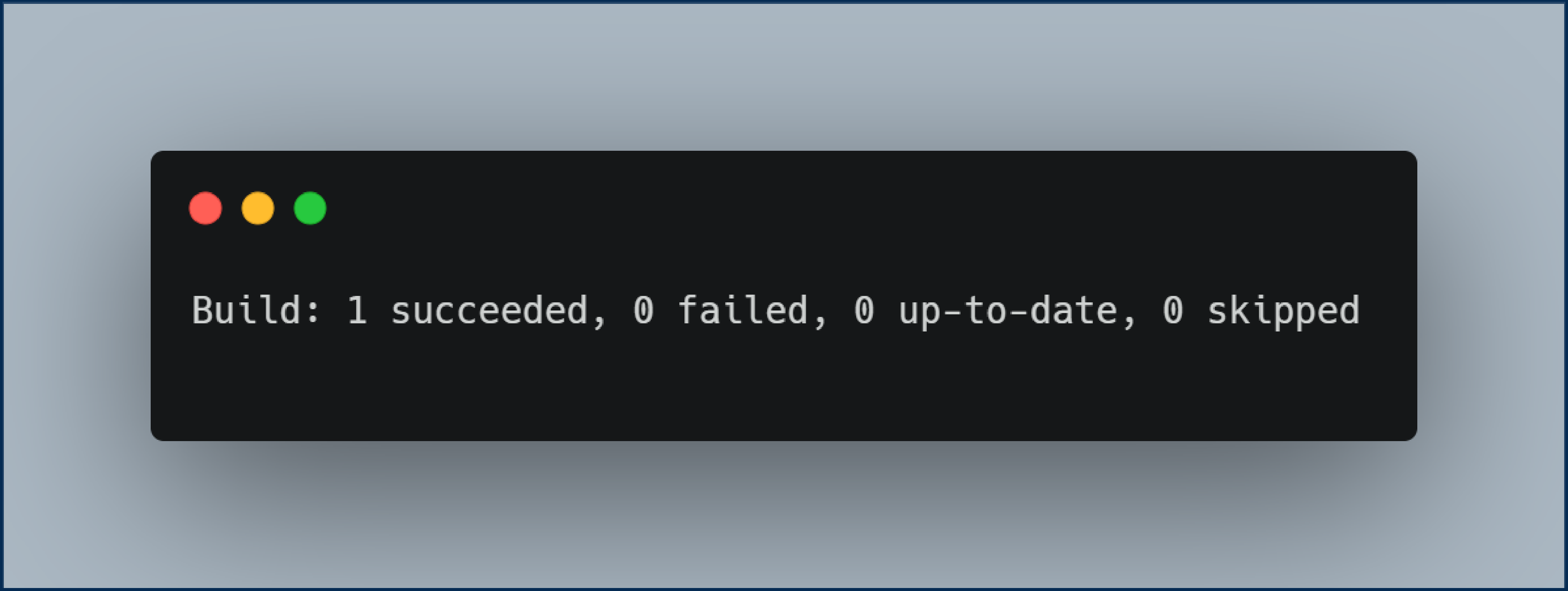
```
warning CS8602: Dereference of a possibly null reference.
```

CHANGE



```
var initials = $"{user.FirstName?[0]} {user.LastName?[0]}";
```

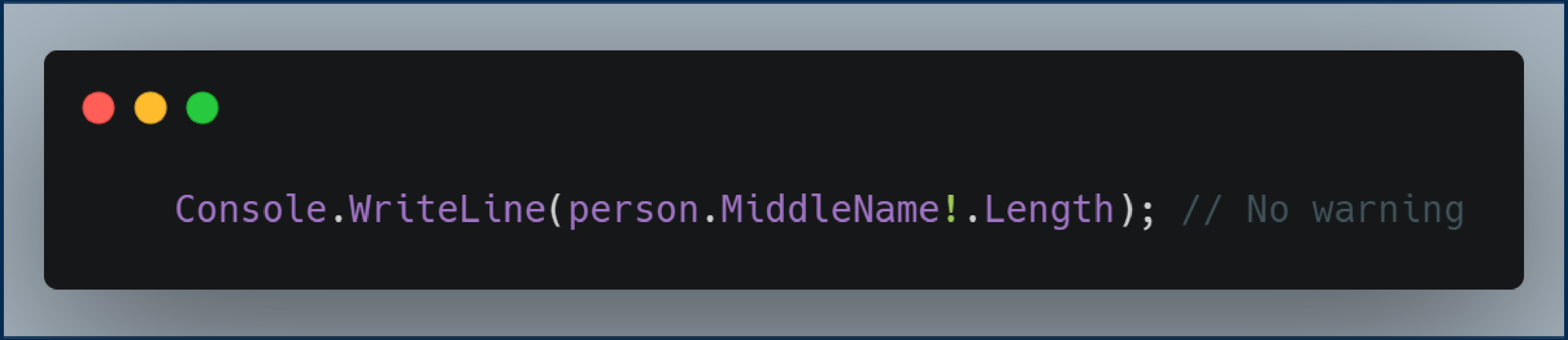
NO WARNINGS



```
Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped
```

What if I know the variable is never null?

! NULL-FORGIVING-OPERATOR



```
Console.WriteLine(person.MiddleName!.Length); // No warning
```

But avoid!

`!` NULL-FORGIVING-OPERATOR

- In **some cases**, the compiler is not able to detect that a nullable value is actually non-nullable.
- **Unit tests** may want to check the behavior of code when a `null` comes through.

COMMON ERRORS

MAKE .NET A NULL-SAFE PLACE

- You write code that expresses your intent
- The compiler enforces that intent