topicus

IMPACT ON SOCIETY

Dennis Spangenberg

Immutable objects in C#

Dennis Spangenberg

- 11 years professional C# .net experience
- Topicus.finance Mortgages
- Stay up-to-date
- Use technique / syntax which im not familair with
- .net Gilde

topicus

topiconf '22

Contents

- What is an immutable object?
- Why do I need immutable objects?
- Immutable objects and records
- Practical fun with immutable objects

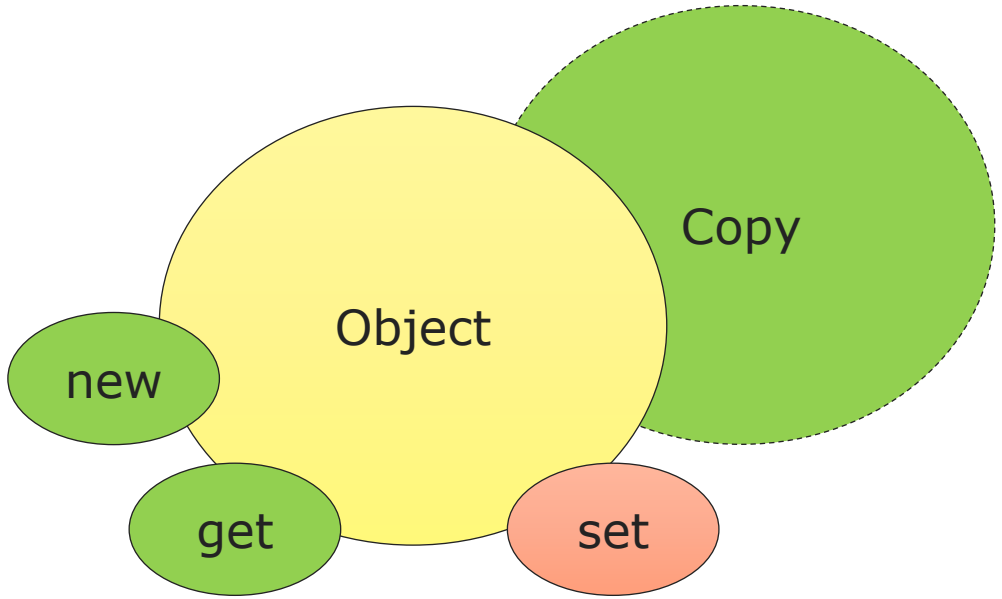
topicus

IMPACT ON SOCIETY

What is an immutable object?

What is an immutable object?

An immutable object (or unchangeable object) is an object whose state cannot be modified after it is created



topicus

topiconf '22

C# example of an immutable object

- C# .net version independent example
- Property only can be set on creation

```
public class ImmutableObject
{
    public readonly string Property;

    5 references
    public ImmutableObject(string property)
    {
        Property = property;
    }
}
```


topicus

topiconf '22

Immutable is read-only

- The compiler will give errors
- It will be less likely to make mistakes

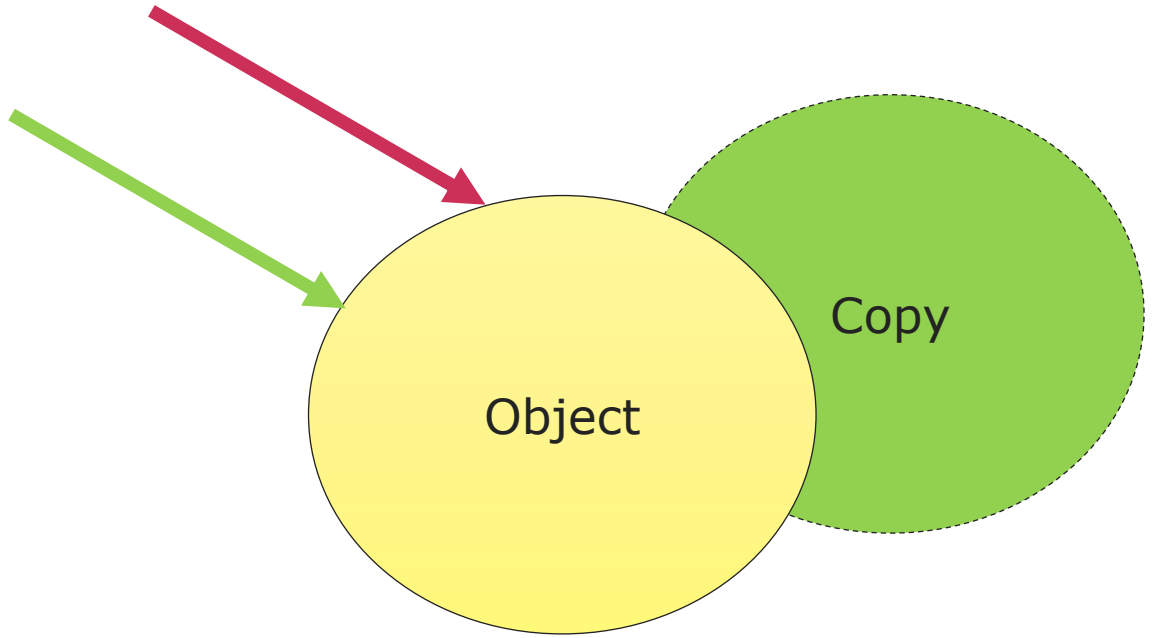
```
var immutableObject = new ImmutableObject( property: "Something");
immutableObject.Property = "Something else";
```


topicus

topiconf '22

Other advantages

- You only can copy or reference an immutable object
- Thread safe



topicus

topiconf '22

Weak vs strong immutability

- Weak: Some properties are still mutable

```
public class WeakImmutableObject
{
    public readonly string Property;

    2 references
    public string Mutable { get; set; }

    5 references
    public WeakImmutableObject(string property, string mutable)
    {
        Mutable = mutable;
        Property = property;
    }
}
```

```
var immutableObject = new WeakImmutableObject( property: "Something",  mutable: "Mutable");
immutableObject.Mutable = "Changed";
immutableObject.Property = "Something else";
```


topicus

topiconf '22

topicus

IMPACT ON SOCIETY

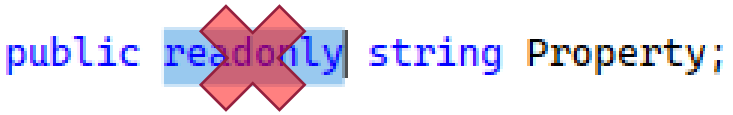
Why do I need immutable objects?

Easy to work with

- Personal experience
- Easy to understand
- Easy to test
- Easy to break…

Common development mistakes

- Making immutable object weak or mutable
- Copying objects instead of referencing



topicus

topiconf '22

Make test to determine if immutable objects stay immutable

```
public class ImmutableTestObject
{
    public readonly string Property;

    //programming error: this should be readonly
    4 references | 0 changes | 0 authors, 0 changes
    public string Immutable { get; private set; }

    2 references | 0 changes | 0 authors, 0 changes
    public ImmutableTestObject(string property, string immutable)
    {
        Property = property;
        Immutable = immutable;
    }

    1 reference | 0 changes | 0 authors, 0 changes
    public ImmutableTestObject SetMutable(string immutable)
    {
        //programming error
        Immutable = immutable;

        return new ImmutableTestObject(Property, immutable);
    }
}
```

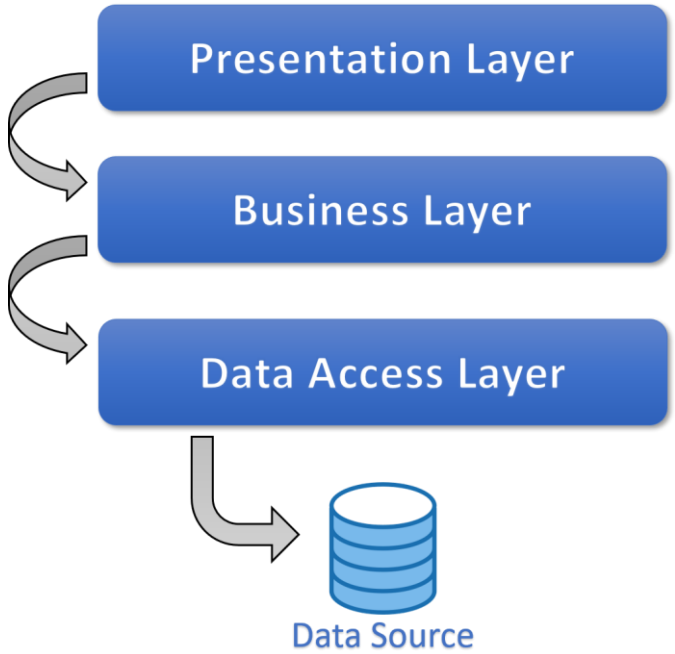
```
public void ImmutableChanged()
{
    var immutable1 = new ImmutableTestObject(property: "Test" , immutable: "Me");
    var immutable2 = immutable1.SetMutable("This");

    Assert.That(immutable1.Immutable, Is.Not.EqualTo(immutable2.Immutable));
}
```


topicus

topiconf '22

Apply to software architecture

- Multitier architecture
- Use immutable object for input and output
- Don’t use immutable object for (database) entities



topicus

topiconf ‘22

topicus

IMPACT ON SOCIETY

Immutable objects and records

Boilerplate code

- **boilerplate code**, or simply **boilerplate**, are sections of code that are repeated in multiple places with little to no variation.

```
public class ImmutableObject
{
    public readonly string Property;

    5 references
    public ImmutableObject(string property)
    {
        Property = property;
    }
}
```


topicus

topiconf '22

Immutable objects in C# 9

- Since C# 9 we have records
- .net 5 and up supported
- Records reduce boilerplate code for immutable object

```
public record ImmutableRecord(string Property);
```

Side by side

```
public class ImmutableObject
{
    public readonly string Property;

    5 references
    public ImmutableObject(string property)
    {
        Property = property;
    }
}
```

```
public record ImmutableRecord(string Property);
```

```
var immutableObject = new ImmutableObject(property: "Something");
immutableObject.Property = "Something else";

var immutableRecord = new ImmutableRecord(Property: "Something");
immutableRecord.Property = "Something else";
```


topicus

topiconf '22

Weather Forecast example

```
public class WeatherForecast
{
    1 reference
    public DateTime Date { get; set; }

    2 references
    public int TemperatureC { get; set; }

    0 references
    public int TemperatureF => 32 + (int)(TemperatureC / 0.5556);

    1 reference
    public string? Summary { get; set; }
}
```

```
public record WeatherForcastRecord(DateTime Date, int TemperatureC, string? Summary)
{
    0 references
    public int TemperatureF => 32 + (int)(TemperatureC / 0.5556);
}
```


topicus

topiconf '22

Less risk on progamming errors with records

- Records take effort to make mutable

```
public record ImmutableRecord(string Property);
```

```
public record MutableRecord(string Property)
{
    public string Property = Property;
}
```

```
var mutableRecord = new MutableRecord( Property: "Something");
mutableRecord.Property = "Something else";
```


topicus

topiconf '22

Weak immutable record

```
var weakImmutableRecord = new WeakImmutableRecord("Jane", "Doe");
weakImmutableRecord.First = "John";
weakImmutableRecord.Last = "Deer";
```

```
public record WeakImmutableRecord(string First, string Last)
{
    1 reference
    public string Last { get; set; } = Last;
}
```


topicus

topiconf '22

topicus

IMPACT ON SOCIETY

Fun with records

Value equality (1/3)

- Records are comparible by default
- On properties the values are compared

```
[Test]
0 references | pangspang, 2 hours ago | 1 author, 1 change
public void RecordEquality()
{
    var immutable1 = new ImmutableRecord( Property: "Test");
    var immutable2 = new ImmutableRecord( Property: "Test");

    Assert.That(immutable1, Is.EqualTo(immutable2));
}
```

```
[Test]
0 references | pangspang, 2 hours ago | 1 author, 1 change
public void RecordReference()
{
    var immutable1 = new ImmutableRecord( Property: "Test");
    var immutable2 = new ImmutableRecord( Property: "Test");

    Assert.That(immutable1, Is.Not.SameAs(immutable2));
}
```

Value equality (2/3)

✓ RecordEquality

```
public void RecordEquality()
{
    var immutable1 = new ImmutableRecord( Property: "Test");
    var immutable2 = new ImmutableRecord( Property: "Test");

    Assert.That(immutable1, Is.EqualTo(immutable2));
}
```

⛔ ObjectEquality

```
public void ObjectEquality()
{
    var immutable1 = new ImmutableObject( property: "Test");
    var immutable2 = new ImmutableObject( property: "Test");

    Assert.That(immutable1, Is.EqualTo(immutable2));
}
```


topicus

topiconf '22

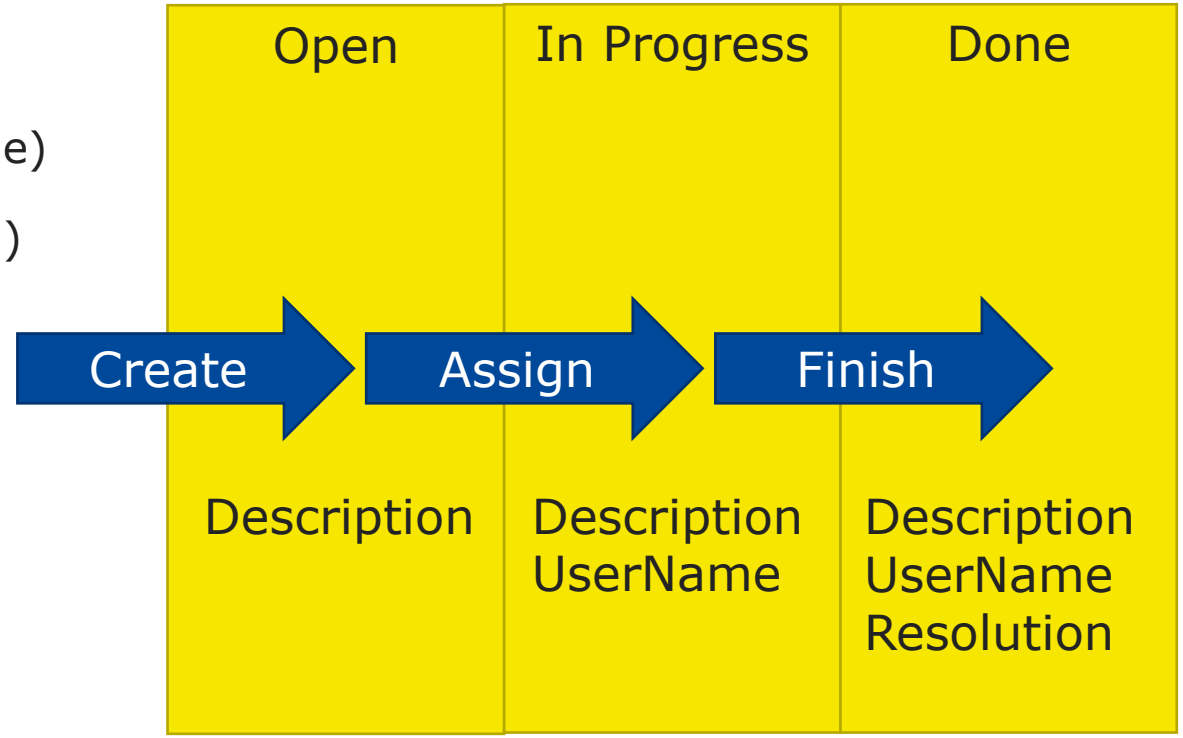
Value equality (3/3)

- Override Equals
- Override Hashcode
- Override '=='
- Implementation changes when adding more properties

```
public class ValueEqualityImmutableObject
{
    public readonly string Property;
    public readonly string OtherProperty;

    2 references | 0 changes | 0 authors, 0 changes
    public ValueEqualityImmutableObject(string property, string otherProperty)
    {
        Property = property;
        OtherProperty = otherProperty;
    }

    0 references | 0 changes | 0 authors, 0 changes
    public override bool Equals(object? obj)
    {
        if (obj == null && obj is not ValueEqualityImmutableObject)
            return false;

        var other = obj as ValueEqualityImmutableObject;

        return Property == other.Property && OtherProperty == other.OtherProperty;
    }

    0 references | 0 changes | 0 authors, 0 changes
    public override int GetHashCode()
    {
        return HashCode.Combine(Property, OtherProperty);
    }
}
```

Using records as statemachine with events

- Kanban board
  - with 3 lanes (Open, In Progress, Done)
  - with 3 actions (Create, Assign, Finish)
- Tickets can be in a lane
  - The lane defines which properties need to be there
- Actions can be applied on tickets



topicus

topiconf '22

Ticket

- Ticket is abstract
- Ticket is inherited
- Ticket implementation have specific properties

```
public abstract record Ticket
{
    2 references | - changes | -authors, -changes
    public record Empty : Ticket;

    4 references | - changes | -authors, -changes
    public record Open(
        string Description
        ) : Ticket;

    7 references | - changes | -authors, -changes
    public record InProgress(
        string Description,
        string UserName
        ) : Ticket;

    2 references | - changes | -authors, -changes
    public record Done(
        string Description,
        string UserName,
        string Resolution
        ) : Ticket;
}
```

Ticketevent

- Events only contain the property needed
- The type of the event gives context

```
public abstract record TicketEvent
{
    2 references | - changes | -authors, -changes
    public record Create(
        string Description
        ) : TicketEvent;

    4 references | - changes | -authors, -changes
    public record Assign(
        string UserName
        ) : TicketEvent;

    2 references | - changes | -authors, -changes
    public record Finish(
        string Resolution
        ) : TicketEvent;
}
```

Move tickets with events

- Switch with pattern matching (C# 9)
- New ticket if state changes
  - destructor is called
  - new record is initialized

```
public Ticket Move(Ticket state, TicketEvent @event) =>
    @event switch
    {
        TicketEvent.Create(var description:string)
            => state is Ticket.Empty
                ? new Ticket.Open(description)
                : state,
        TicketEvent.Assign assign
            => Assign(state, assign),
        TicketEvent.Finish(var resolution:string)
```

Assign

- 2 actions
  - newly in progress
  - reassign to other user
- Reassign works with “with”

  Copy of original, only changes the property

```
public Ticket Assign(Ticket state, TicketEvent.Assign assignEvent) =>
    state switch
    {
        Ticket.Open(var description :string) =>
            new Ticket.InProgress(description, assignEvent.UserName),
        Ticket.InProgress inProgress =>
            inProgress with { UserName = assignEvent.UserName },
```

The end

- Wrap up:
  - What are immutable objects
  - How can I use them in my .net projects
  - With some cool examples ☺
- Code examples: https://github.com/pangspang/ImmutableObjectInCSharp
- Join the #net-gilde (slack)
- I’m Dennis Spangenberg, any questions?

topicus

topiconf ‘22