Stage 1: Naive Models (Illegal States Problem)

Branch: `demo/stage-1-naive-csharp` | **Time:** 5-15 min | **Goal:** Show boolean flags allow illegal states

F# Code (Library.fs)

```
namespace CustomerLiveDomain

type Customer = {
    Id: string
    IsEligible: bool
    IsRegistered: bool
}
```

C# Code (Program.cs - Key Parts)

```
var john = new Customer { Id = "John", IsEligible = true, IsRegistered = true };
var grinch = new Customer { Id = "Grinch", IsEligible = true, IsRegistered = false
}; // ❌ ILLEGAL!

static decimal CalculateTotal(Customer customer, decimal spend)
{
    // Correct version:
    if (customer.IsEligible && customer.IsRegistered && spend >= 100)
        return spend * 0.9m;
    return spend;

    // Buggy version (forgot IsRegistered check):
    // if (customer.IsEligible && spend >= 100)
    //     return spend * 0.9m;
}
```

The Problem

2 booleans = 4 states, but only 3 are valid:

- `false, false` → ☑ Guest
- `false, true` → ☑ Registered (not eligible)
- `true, true` → ☑ Standard (eligible)
- `true, false` → ❌ **ILLEGAL** (eligible but not registered)

Grinch example: Can create `IsEligible=true, IsRegistered=false` - compiler doesn't prevent it!

Demo Script

1. Show F# record (immutable but allows illegal state)

2. Create john (valid) and grinch (illegal state)
3. Run with **correct** logic → both work correctly
4. Switch to **buggy** logic (comment out `IsRegistered` check)
5. Run again → Grinch gets discount! ❌

Key Talking Points

- "F# immutability is nice but doesn't solve domain problems"
- "Easy to forget one boolean check - happens all the time!"
- "Compiler can't help - it accepts the illegal state"
- "Solution? Make illegal states unrepresentable!"

Transition to Stage 2

"F# discriminated unions let us model **actual states** (Registered vs Guest) instead of boolean combinations. Let's refactor..."

Recovery

```
git reset --hard
git checkout demo/stage-1-naive-csharp
```