Stage 2: First Discriminated Union (DU Prevents Illegal States)

Branch: `demo/stage-2-first-du` | **Time:** 15-20 min | **Goal:** DUs make registration explicit

F# Code (Library.fs)

```fsharp
namespace CustomerLiveDomain

type RegisteredCustomer = {
    Id: string
    IsEligible: bool      // Only registered customers CAN have this field
}

type UnregisteredCustomer = {
    Id: string            // No IsEligible field!
}

type Customer =
    | Registered of RegisteredCustomer
    | Guest of UnregisteredCustomer
```

C# Code (Program.cs)

```csharp
using static CustomerLiveDomain.Customer;

var john = NewRegistered(new RegisteredCustomer { Id = "John", IsEligible = true
});
var sarah = NewGuest(new UnregisteredCustomer { Id = "Sarah" });

// Try to create illegal state:
// var grinch = NewGuest(new UnregisteredCustomer { Id = "Grinch", IsEligible =
true });
// ❌ COMPILER ERROR: UnregisteredCustomer has no IsEligible property!

static decimal CalculateTotal(Customer customer, decimal spend)
{
    // Linear approach (works with .NET Framework):
    if (customer.IsRegistered)
    {
        var registered = (Customer.Registered)customer;
        if (registered.Item.IsEligible && spend >= 100)
            return spend * 0.9m;
    }
    return spend;

    // Pattern matching alternative (modern C#):
```

```
    // var discount = customer switch
    // {
    //     Registered c when c.Item.IsEligible && spend >= 100 => spend * 0.1m,
    //     _ => 0m
    // };
    // return spend - discount;
}
```

Live Refactoring Steps

1. **Create record types:** `RegisteredCustomer` (with IsEligible), `UnregisteredCustomer` (without)
2. **Create DU:** `type Customer = Registered of ... | Guest of ...`
3. **Update C#:** Use `NewRegistered()` and `NewGuest()` factory methods
4. **Attempt illegal state:** Try adding `IsEligible` to Guest → compiler error!

What F# Generates for C#

```
Customer.NewRegistered(RegisteredCustomer)  // Factory method
Customer.NewGuest(UnregisteredCustomer)     // Factory method
customer.IsRegistered                       // Boolean property
customer.IsGuest                            // Boolean property
```

Key Improvements Over Stage 1

☑ **Can't create eligible guest** - `UnregisteredCustomer` has NO `IsEligible` field
☑ **Domain language clearer** - "Registered" vs "Guest" explicit
☑ **Type system prevents bugs** - Illegal state unrepresentable
⚠ **Still has boolean** - `IsEligible` on `RegisteredCustomer` (Stage 3 fixes this)

Key Talking Points

- "DU says: Customer is **EITHER** Registered **OR** Guest, never both"
- "Type system prevents illegal state at **compile time**, not runtime"
- "F# generates C# interop code automatically - seamless!"
- "But we still have a boolean... Stage 3 will eliminate it"

Transition to Stage 3

"We've eliminated one illegal state, but `IsEligible` boolean can still be forgotten. Let's make eligibility tiers **explicit types** instead of a flag!"

Recovery

```
git reset --hard; git checkout demo/stage-2-first-du
```