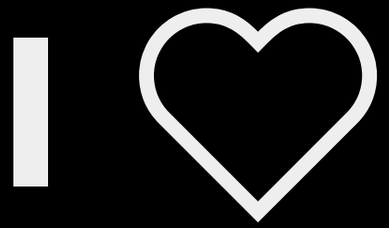
Write Your Own Domain Specific Language with F#

Mikhail Smal

https://smal.dev

@ DevDays Europe 2019

My name is Mikhail and

I  F#

What about you?

F#

- Functional-first language
- ADT + pattern matching
- Cross-platform thx to .NET Core
- Compiles to JS thx to Fable

Why even bother?

Reading the code

We want to understand the intention

Uncertainity

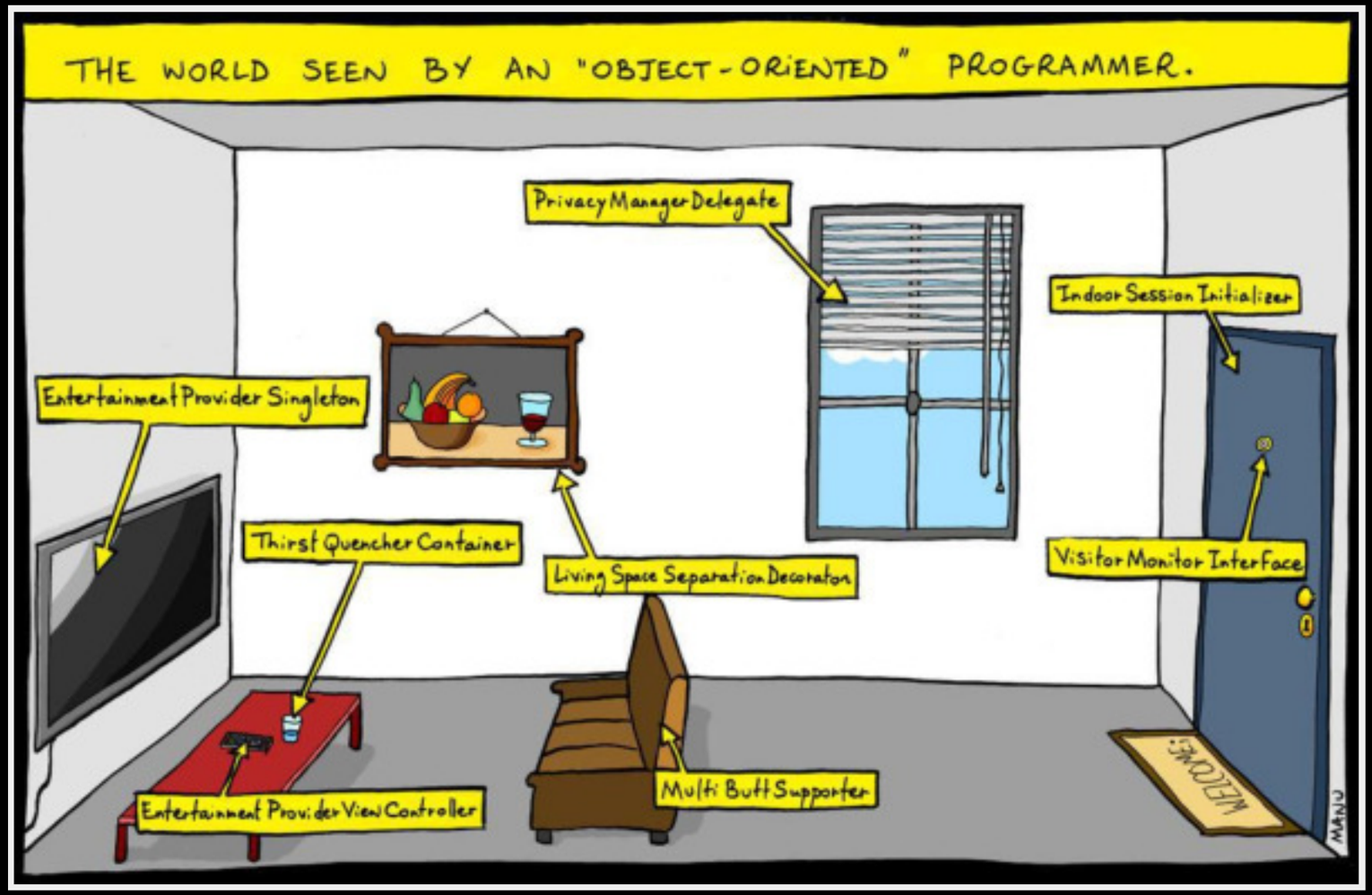
```
1: public class OrderController : Controller
2: {
3:     public IActionResult Get()
4:     {
5:         // Null? Exception?
6:         List<Order> orders = _ordersService.GetAllOrders();
7:         return Json(orders);
8:     }
9: }
```

Nulls

```
 1: public void CreateOrder(Order order, string userId)
 2: {
 3:     if (order == null)
 4:     {
 5:         throw new ArgumentNullException(nameof(order));
 6:     }
 7:     if (userId == null)
 8:     {
 9:         throw new ArgumentNullException(nameof(userId));
10:     }
11:     // Business logic. Finally...
12: }
13:
14:
```

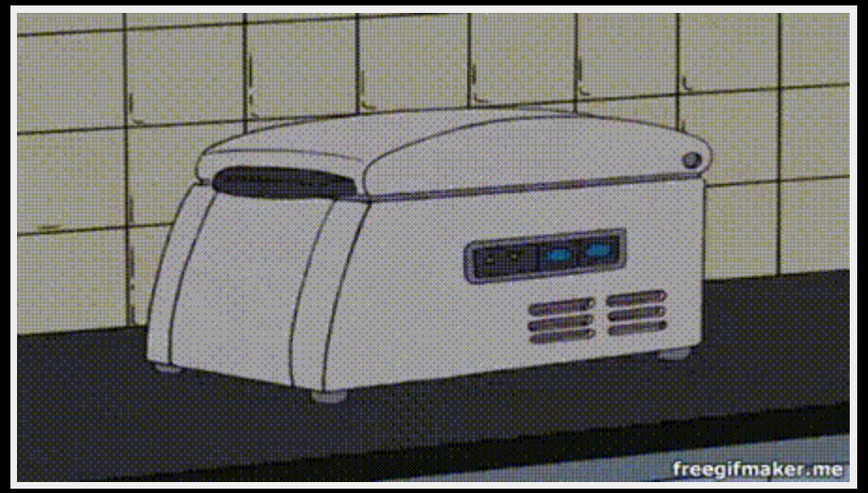
Keywords noise

- public
- static
- readonly
- void
- class
- struct



http://bonkersworld.net/object-world

```
1: AbstractToastHeaterGeneratorFactoryInterfaceImplementer
2:     abstractToastHeaterGeneratorFactoryInterfaceImplementer =
3:     new AbstractToastHeaterGeneratorFactoryInterfaceImplementer(
4:         AbstractToastHeaterGeneratorFactoryInterfaceImplementer.DEFAULT_PARAMS, 0, NULL);
```



```
{
    "thankYou": "Thank you",
    "supportWillContactYou": "Our support team will contact you shortly",
    "welcomeToSurvey": "Welcome to ticket survey",
    "send": "Send",
}

```

Different languages

- float?
- integer?
- Guid?
- string?


Olya
@w0lya
Follow
Replying to @w0lya @lenadroid
And some programming languages (like #fsharp ;)) are so expressive that you can just write pseudo code instead of comments. Then uncomment, tweak a bit for everything to compile, and you're done with implementation, things just work!
7:38 AM - 23 Feb 2019

Modeling with types

```fsharp
type CardType = // 'OR' type
    | Visa
    | Mastercard
type CheckNumber = CheckNumber of int
type CardNumber = CardNumber of string

type CreditCardInfo = { // 'AND' type (record)
    CardType : CardType
    CardNumber : CardNumber
}

type PaymentMethod =
    | Cash
    | Check of CheckNumber
    | Card of CreditCardInfo

type PaymentAmount = PaymentAmount of decimal
type Currency = EUR | USD

type Payment = {
    Amount : PaymentAmount
    Currency : Currency
    Method : PaymentMethod
}
```

```fsharp
namespace Domain

type [<Measure>] g
type [<Measure>] inch
type Gramms = Gramms of int<g>
type Inches = Inches of float<inch>

type GadgetName = GadgetName of string

// 5 characters starting with T
type PhoneCode = PhoneCode of string
// 10000 < code < 100000
type TabletCode = TabletCode of int

type GadgetCode =
    | PhoneCode of PhoneCode
    | TabletCode of TabletCode

type Gadget = {
    Code : GadgetCode
    Name : GadgetName
    Weight : Gramms
    ScreenSize : Inches
}
```

```
type UnvalidatedOrder = {
    ...
    ShippingAddress : UnvalidatedAddress
    ...
}

type ValidatedOrder = {
    ...
    ShippingAddress : ValidatedAddress
    ...
}

type AddressValidationService = UnvalidatedAddress -> ValidatedAddress option

type Option<'a> =
    | Some of 'a
    | None
```

```
type Result<'Success,'Failure> =
    | Ok of 'Success
    | Error of 'Failure

type PaymentError =
    | CardTypeNotRecognized
    | PaymentRejected
    | PaymentProviderOffline

type PayInvoice = UnpaidInvoice -> Payment -> Result<PaidInvoice,PaymentError>

```

```
1: type ValidateOrder = UnvalidatedOrder -> Result<ValidatedOrder,ValidationError list>
2: and ValidationError = {
3:     FieldName : string
4:     ErrorDescription : string
5: }
```

```
namespace Cart

type Item = ...
type ActiveCartData = { UnpaidItems: Item list }
type PaidCartData = { PaidItems: Item list; Payment: Payment }
type ShoppingCart =
    | EmptyCart // no data
    | ActiveCart of ActiveCartData
    | PaidCart of PaidCartData

let addItem cart item = // ShoppingCart -> Item -> ShoppingCart
    match cart with
    | EmptyCart ->
        // create a new active cart with one item
        ActiveCart { UnpaidItems = [item] }
    | ActiveCart { UnpaidItems = existingItems } ->
        // create a new ActiveCart with the item added
        ActiveCart { UnpaidItems = item :: existingItems }
    | PaidCart _ ->
        // ignore
        cart
```

Functions, not classes

```
let validateOrder unvalidateOrder =
    ...
    Ok validatedOrder

let priceOrder validatedOrder =
    ...
    Ok pricedOrder

let acknowledgeOrder pricedOrder =
    ...
    acknowledgement

let createEvents aknowledgement =
    ...
    events

let placeOrder unvalidatedOrder =
    unvalidatedOrder
    |> validateOrder
    |> priceOrder
    |> acknowledgeOrder
    |> createEvents
```

```
type ValidateOrder = UnvalidatedOrder -> Result<ValidatedOrder, ValidationError>
type PriceOrder = ValidatedOrder -> Result<PricedOrder, PricingError>
type AcknowledgeOrder = PricedOrder -> OrderAcknowledgmentSent option
type CreateEvents = PricedOrder -> OrderAcknowledgmentSent option -> PlaceOrderEvent list

let validateOrder : ValidateOrder =
    fun unvalidatedOrder ->
        ...

let priceOrder : PriceOrder =
    fun validatedOrder ->
        ...

let acknowledgeOrder : AcknowledgeOrder =
    fun pricedOrder ->
        ...

let createEvents : CreateEvents =
    fun pricedOrder aknowledgement ->
        ...
```

```
1: let placeOrder unvalidatedOrder =
2:     unvalidatedOrder
3:     |> validateOrderAdapted
4:     |> Result.bind priceOrderAdapted
5:     |> Result.map acknowledgeOrder
6:     |> Result.map createEvents
```

```
1: type PlaceOrderWorkflow = UnvalidatedOrder -> PlaceOrderEvent list
2:
3: let placeOrder : PlaceOrderWorkflow =
4:     fun unvalidatedOrder ->
5:         let validatedOrder = unvalidatedOrder |> validateOrder
6:         let pricedOrder = validatedOrder |> priceOrder
7:         let acknowledgementOption = pricedOrder |> acknowledgeOrder
8:         let events = createEvents pricedOrder acknowledgementOption
9:         events
```

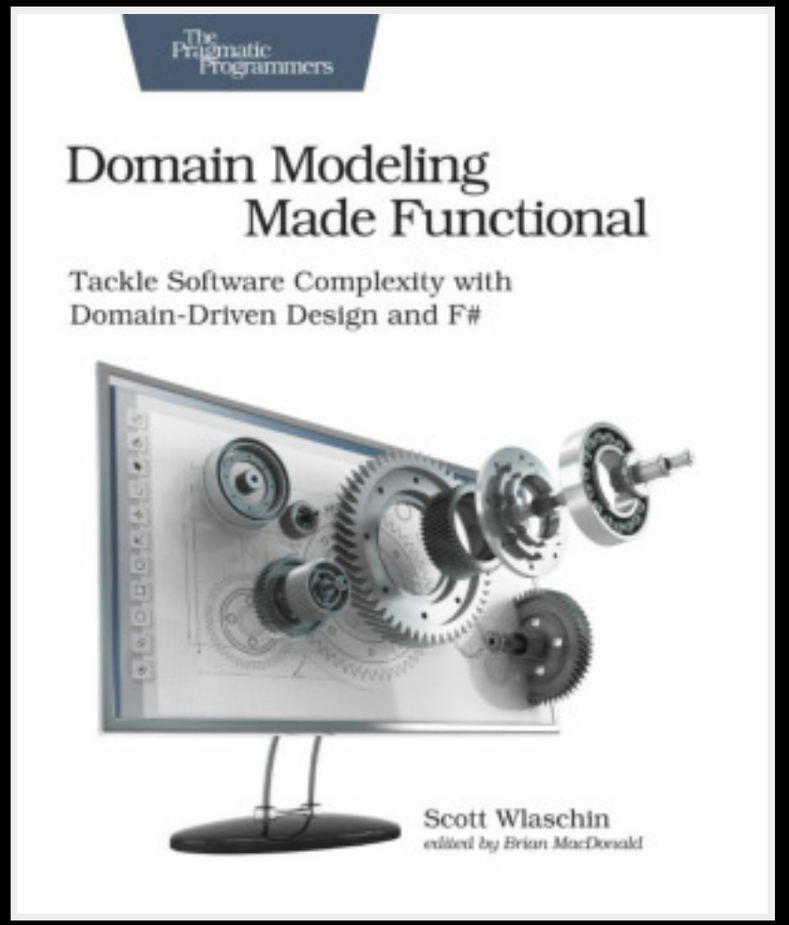
```
let checkProductCodeExists unvalidatedOrder =
    ...

let checkAddressExists unvalidatedOrder =
    ...

let validateOrder checkProductCodeExistsFun checkAddressExistsFun unvalidatedService =
    ...

let placeOrder unvalidatedOrder =
    let validateOrder = validateOrder checkProductCodeExists checkAddressExists
    unvlidatedOrder
    |> validatedOrder
    |> ...
    |> ...
```

Domain Modeling Made Functional



Scott Wlaschin

Why F# is the best enterprise language

https://fsharpforfunandprofit.com/posts/fsharp-is-the-best-enterprise-language/

https://safe-stack.github.io/

Thank you

Ačiū

Mikhail Smal

https://smal.dev