Building Angular-powered Templates

Matt Honeycutt

@matthoneycutt | http://trycatchfail.com/strongly-typed-ajs

Our Markup

```
<div class="form-group has-feedback">
    <label class="control-label" for="Name">Name</label>
    <input required ng-model="vm.customer.Name"
           class="form-control" name="Name" type="text" placeholder="Full name (ex: John Smith)...">
</div>

<div class="form-group has-feedback">
    <label class="control-label" for="WorkEmail">Work Email</label>
    <input required ng-model="vm.customer.WorkEmail"
           class="form-control" name="WorkEmail" type="email" placeholder="user@domain.com...">
</div>

<div class="form-group has-feedback">
    <label class="control-label" for="HomeEmail">Home Email</label>
    <input ng-model="vm.customer.HomeEmail"
           class="form-control" name="HomeEmail" type="email" placeholder="user@domain.com...">
</div>
```

Our Markup

```
<div class="form-group has-feedback">
    <label class="control-label" for="Name">Name</label>
    <input required ng-model="@customer.ExpressionFor(x => x.Name)"
           class="form-control" name="Name" type="text" placeholder="Full name (ex: John Smith)...">
</div>

<div class="form-group has-feedback">
    <label class="control-label" for="WorkEmail">Work Email</label>
    <input required ng-model="@customer.ExpressionFor(x => x.WorkEmail)"
           class="form-control" name="WorkEmail" type="email" placeholder="user@domain.com...">
</div>

<div class="form-group has-feedback">
    <label class="control-label" for="HomeEmail">Home Email</label>
    <input ng-model="@customer.ExpressionFor(x => x.HomeEmail)"
           class="form-control" name="HomeEmail" type="email" placeholder="user@domain.com...">
</div>
```

Our Markup

```
@customer.FormGroupFor(x => x.Name)

@customer.FormGroupFor(x => x.WorkEmail)

@customer.FormGroupFor(x => x.HomeEmail)

@customer.FormGroupFor(x => x.WorkPhone)

@customer.FormGroupFor(x => x.HomePhone)

@customer.FormGroupFor(x => x.WorkAddress)

@customer.FormGroupFor(x => x.HomeAddress)
```

Validation

```
@customer.FormGroupFor(x => x.Name)
```

Full Name

Full Name (ex: John Doe)...

Full Name

Full Name (ex: John Doe)...

Full Name

John Doe

```
@customer.FormGroupFor(x => x.HomeAddress)
```

(Don't!) Repeat Yourself

```
<form novalidate
      name="vm.form"
      ng-submit="vm.form.$valid && vm.add()">

    <!-- Snip -->

    @customer.FormGroupFor(x => x.Name)

    @customer.FormGroupFor(x => x.WorkEmail)

    @customer.FormGroupFor(x => x.HomeEmail)

    @customer.FormGroupFor(x => x.WorkPhone)

    @customer.FormGroupFor(x => x.HomePhone)

    @customer.FormGroupFor(x => x.WorkAddress)

    @customer.FormGroupFor(x => x.HomeAddress)

    <!-- Snip -->

</form>
```

Overview

Crash course: reflection, generics, and expressions

Building the FormForModel helper

Dealing with hidden fields

Applying FormForModel *everywhere*

Reflection, Generics, and Lambdas, Oh My!

Reflection

+

Generics

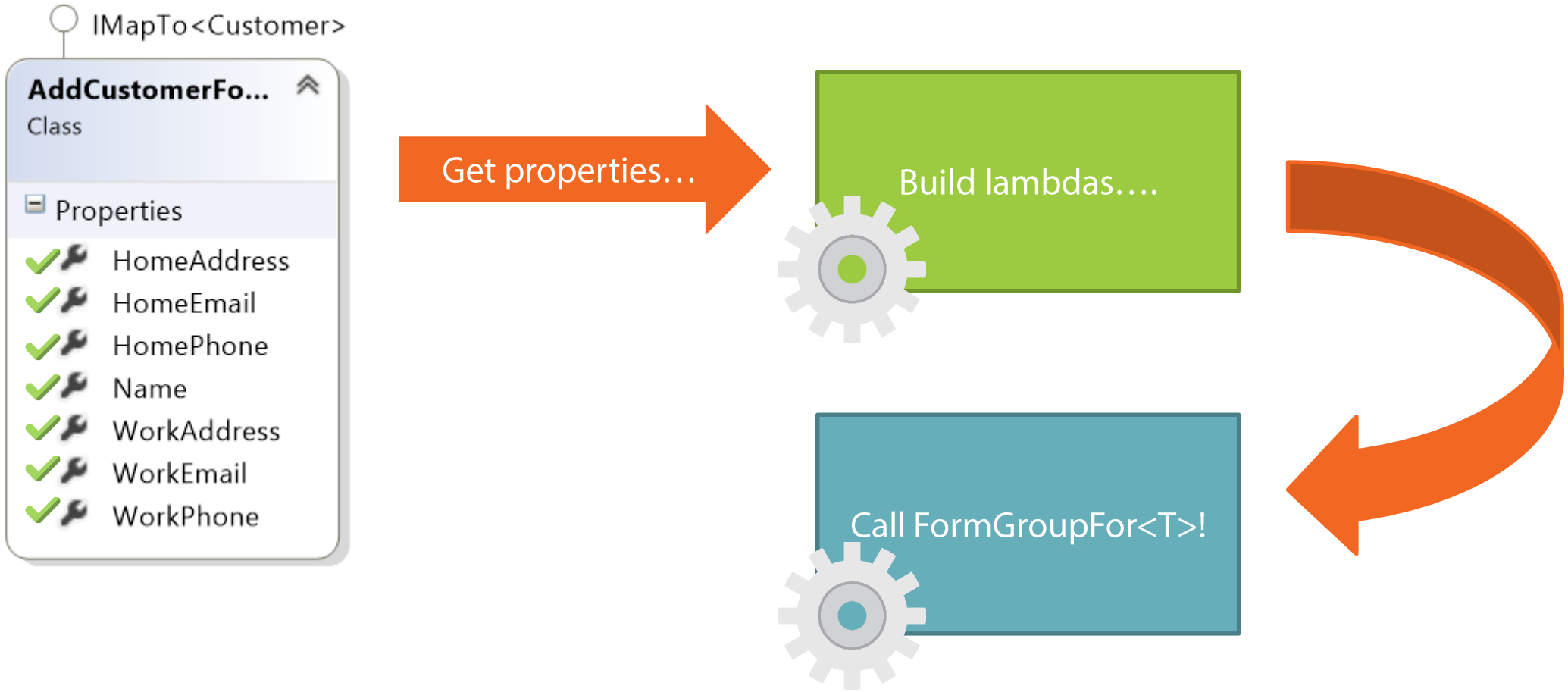
+

Lambda expressions

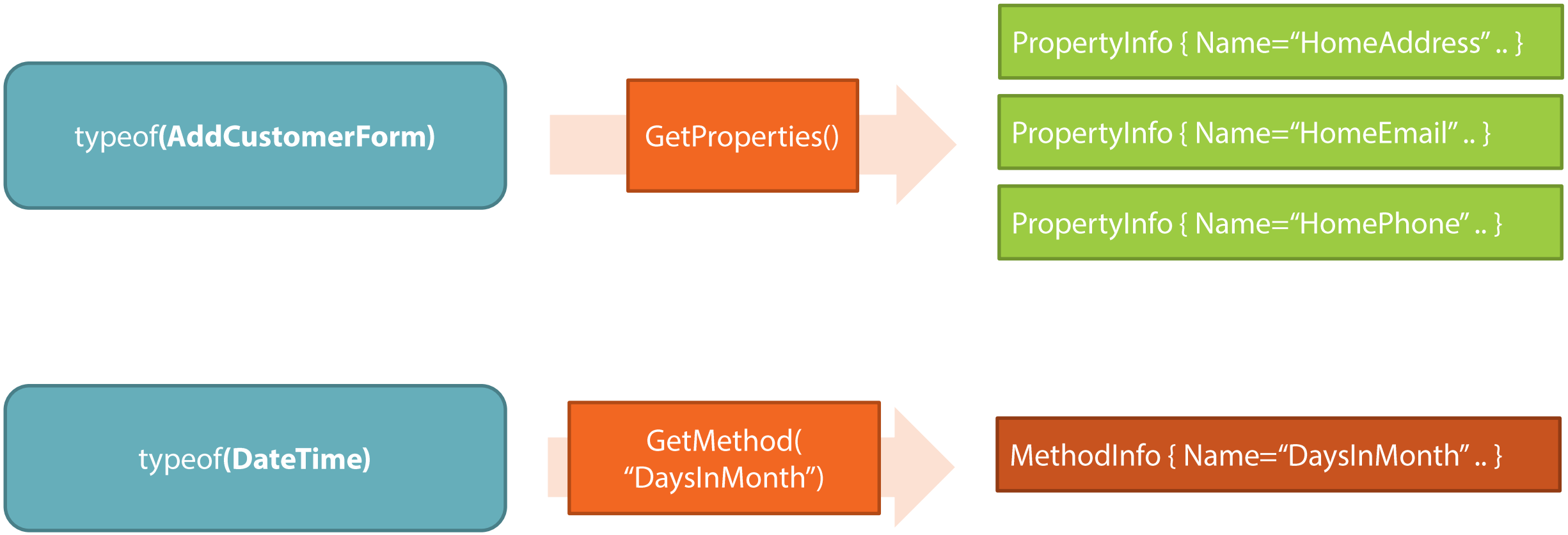
=

DANGER!

Reflection, Generics, and Lambdas, Oh My!



Reflection



Reflection

MethodInfo { Name="DaysInMonth" .. }

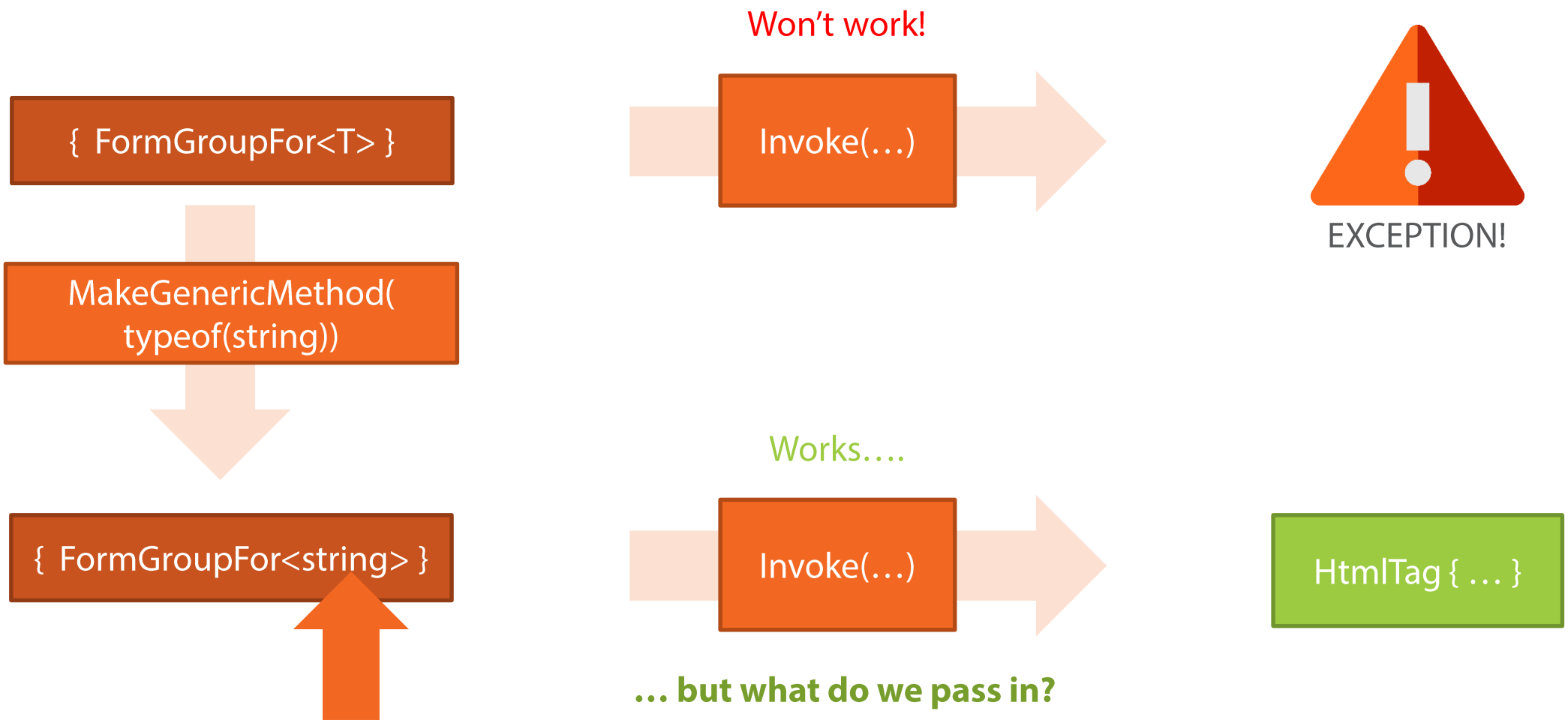
Reflection

Reflection and Generics

Reflection and Generics

{ FormGroupFor<T> }

Reflection and Generics



Lambda Expressions

So far:

```
@customer.FormGroupFor(x => x.Name)
```

AngularModelHelper<AddCustomerForm>.FormGroupFor<string>(…)

Now:

AngularModelHelper<Type???>.FormGroupFor<Type???>(…)

The Expression Helper Class

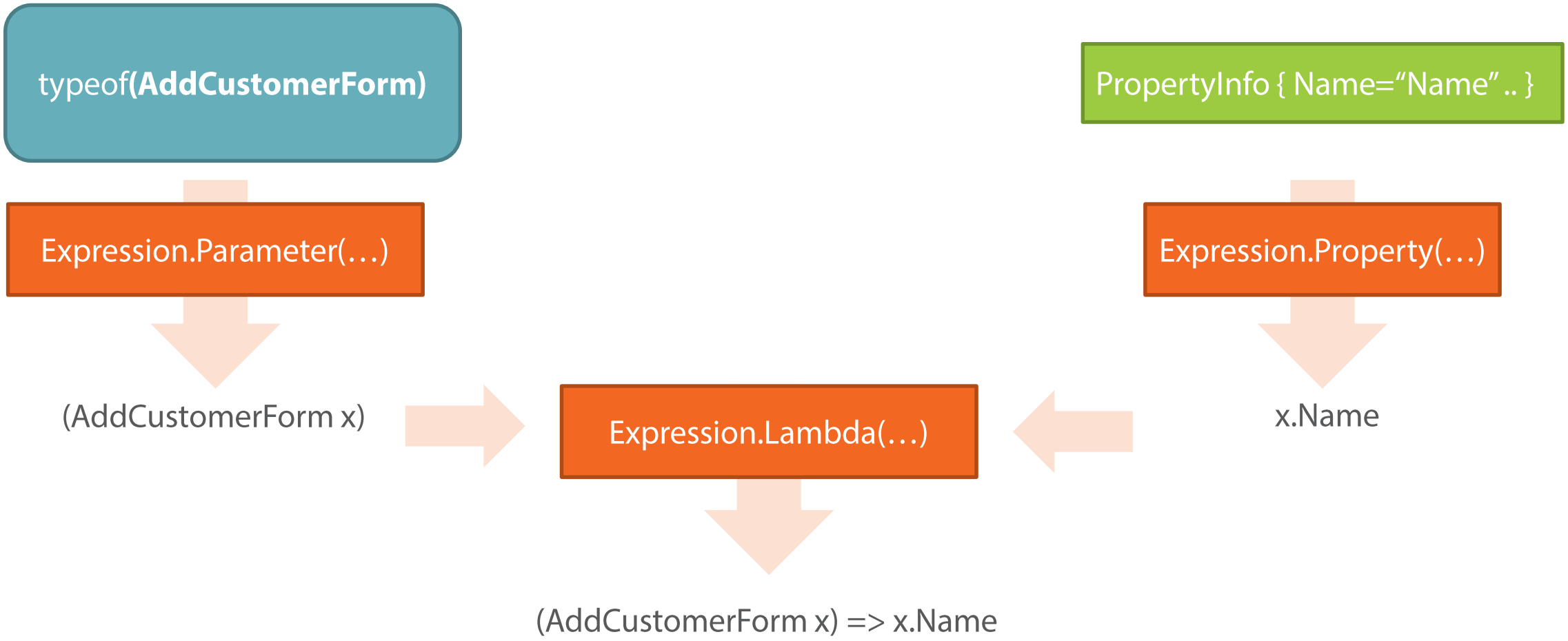
Expression
Abstract Class

Methods

- Accept
- Add (+ 1 overload)
- AddAssign (+ 2 overloads)
- AddAssignChecked (+ 2 overloads)
- AddChecked (+ 1 overload)
- And (+ 1 overload)
- AndAlso (+ 1 overload)
- AndAssign (+ 2 overloads)
- ArrayAccess (+ 1 overload)
- ArrayIndex (+ 2 overloads)
- ArrayLength
- Assign

NOTE: There are several dozen more static methods on the Expression class!

The Expression Helper Class

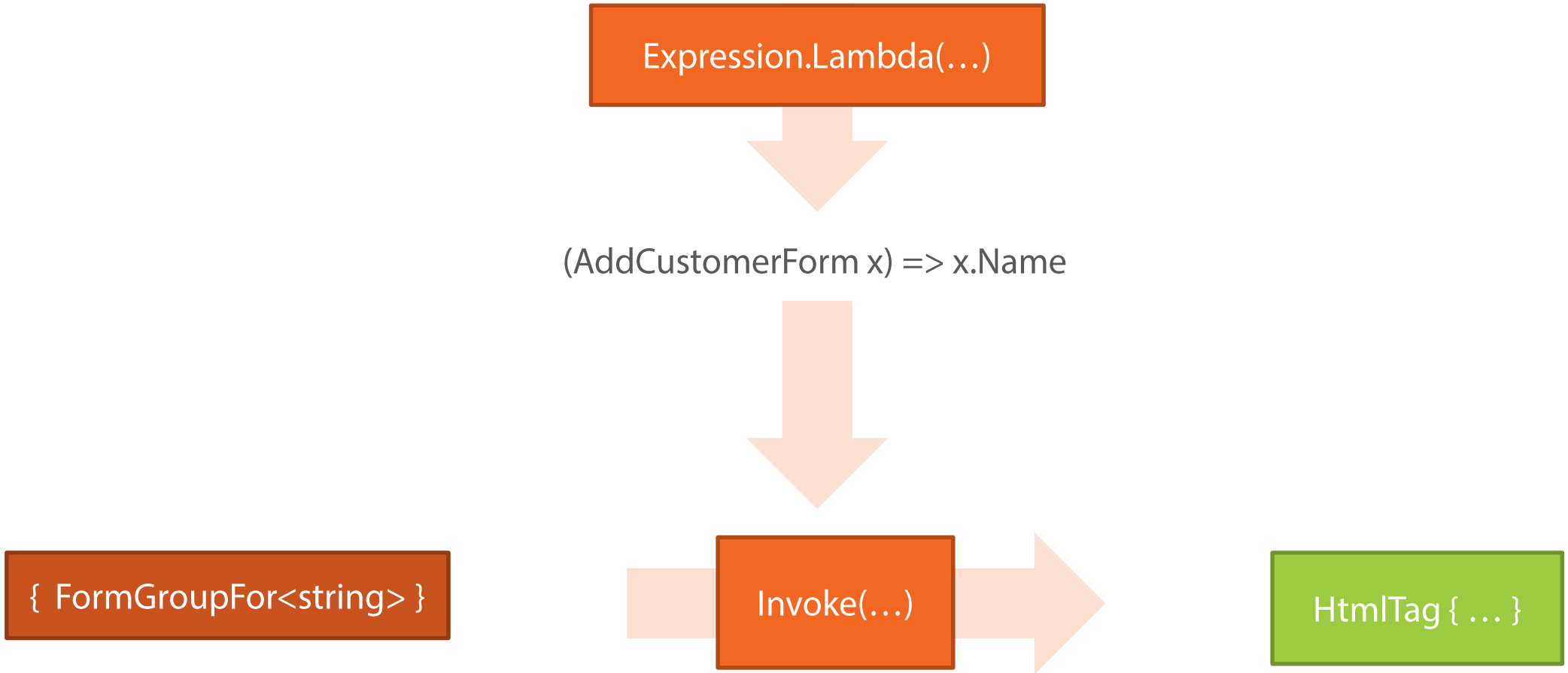


The Expression Helper Class

Expression.Lambda(…)

(AddCustomerForm x) => x.Name

The Expression Helper Class



Reflecting on Reflection and Lambdas

typeof(T) – retrieves type information

GetProperties() – retrieves information about properties

GetMethod(…) – retrieves method info (which can be used to call a method!)

MakeGenericMethod(…) – "closes" type parameters to make a generic method callable

Reflecting on Reflection and Lambdas

Expression.Parameter – create the parameter portion

Expression.Property – create the property-accessor

Expression.Lambda – combine the pieces into an actual lambda expression

Building the FromForModel Helper

What We've Accomplished

```
<form novalidate
      name="vm.form"
      ng-submit="vm.form.$valid && vm.add()">

    <!-- Snip -->

    @customer.FormGroupFor(x => x.Name)

    @customer.FormGroupFor(x => x.WorkEmail)

    @customer.FormGroupFor(x => x.HomeEmail)

    @customer.FormGroupFor(x => x.WorkPhone)

    @customer.FormGroupFor(x => x.HomePhone)

    @customer.FormGroupFor(x => x.WorkAddress)

    @customer.FormGroupFor(x => x.HomeAddress)

    <!-- Snip -->

</form>
```

What We've Accomplished

```
<form novalidate
      name="vm.form"
      ng-submit="vm.form.$valid && vm.add()">

    <!-- Snip -->

    @Html.Angular().FormForModel("vm.customer")


    <!-- Snip -->

</form>
```

What We've Accomplished

```
public class AddCustomerForm : IMapTo<Customer>
{

    public string Name { get; set; }

    public string WorkEmail { get; set; }

    public string HomeEmail { get; set; }

    public string WorkPhone { get; set; }

    public string HomePhone { get; set; }

    public string WorkAddress { get; set; }

    public string HomeAddress { get; set; }
}
```

What We've Accomplished

```
public class AddCustomerForm : IMapTo<Customer>
{
    [Required, Display(Name = "Full Name",
        Prompt = "Full Name (ex: John Doe)...")]
    public string Name { get; set; }

    [Required, DataType(DataType.EmailAddress)]
    public string WorkEmail { get; set; }

    [DataType(DataType.EmailAddress)]
    public string HomeEmail { get; set; }

    [Required, DataType(DataType.PhoneNumber)]
    public string WorkPhone { get; set; }

    [DataType(DataType.PhoneNumber)]
    public string HomePhone { get; set; }

    [Required, DataType(DataType.MultilineText)]
    public string WorkAddress { get; set; }

    [DataType(DataType.MultilineText)]
    public string HomeAddress { get; set; }
}
```

Full Name

Full Name (ex: John Doe)...

Work Email

Work Email...

Home Email

Home Email...

Work Phone

Work Phone...

Home Phone

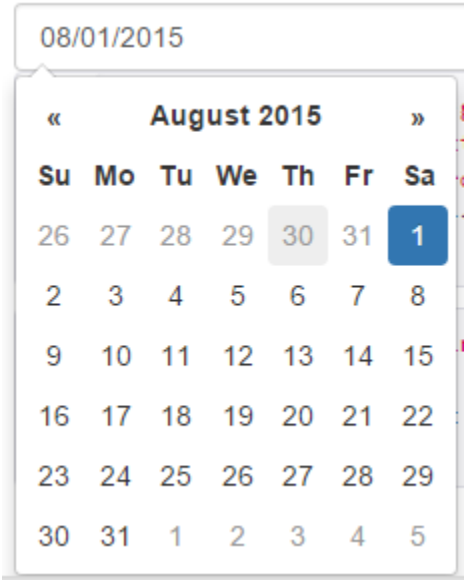
Home Phone...

Work Address

Work Address...

This Isn't the End!

```
[DataType(DataType.Date)]
public DateTime OrderDate { get; set; }
```




```
[Range(1, 99)]
public int Age { get; set; }
```

Up Next…

Building helpers for
reusable directives!