# Modeller, datainmatning och validering



# Upphovsrätt för detta verk

Detta verk är framtaget i anslutning till kursen ASP.NET MVC vid Linnéuniversitetet.

**Du får använda detta verk så här:**



**Det betyder att du i icke-kommersiella syften får:**

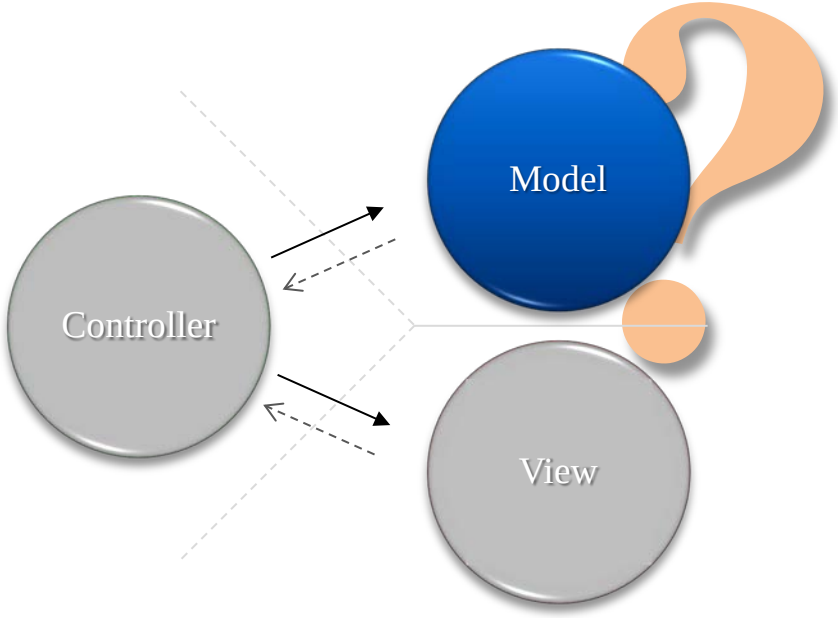
- kopiera hela eller delar av innehållet
- sprida hela eller delar av innehållet
- visa hela eller delar av innehållet offentligt och digitalt
- konvertera innehållet till annat format
- du får även göra om innehållet

Om du förändrar innehållet så ta inte med Linnéuniversitetets logotyp och symbol samt ikoner och fotografier i din nya version!

Vid all användning måste du ange källan: ”Linnéuniversitetet – ASP.NET MVC” och en länk till https://coursepress.lnu.se/kurs/aspnet-mvc och till Creative Common-licensen här ovan.

# Vad är en modell?

- ✓ I MVC används modell för att beteckna dataobjektet som skickas från controllern till vyn.
- ✓ ASP.NET MVC har inga krav på att en typ som representerar en modell ska ärva från en specifik basklass.
- ✓ En modell har i regel publika egenskaper.
  - En modells egenskaper kan renderas av vyn genom att använda *HTML Helpers*.
  - Formulärdata kan automatiskt av en controller bindas till en modells egenskaper.
  - Genom att dekorera en egenskap med attribut kan validering implementeras på server och klient.



# En enkel klass är modellen

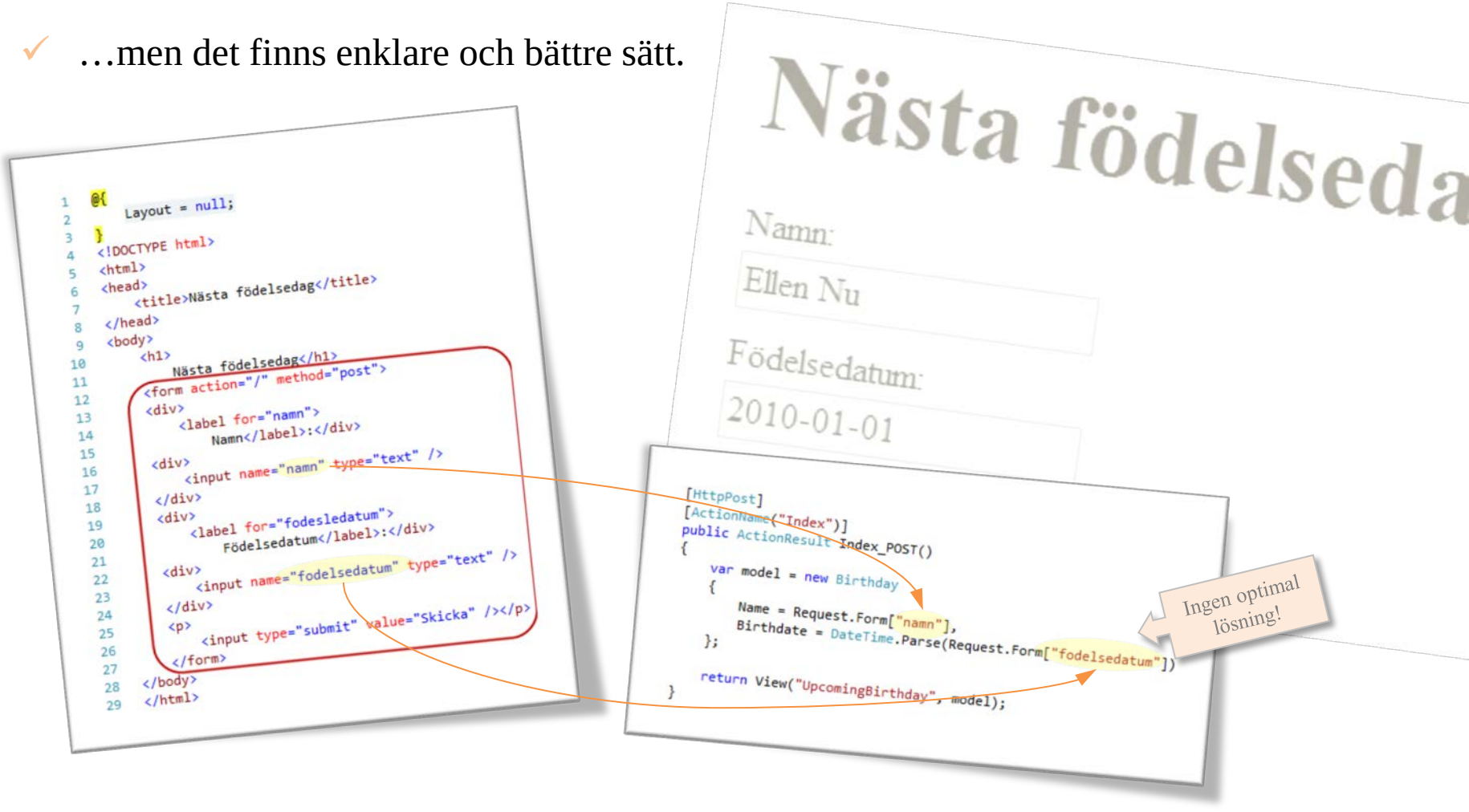
- En instans av klassen `Birthday`, en helt vanlig klass, är ett exempel på en modell som kan skickas från en controller till en vy.
- Egenskaperna `Birthdate` och `Name` används för initiera objektet.
- `Age`, `DaysUntilNextBirthday` och `NextBirthdayDate` är *read-only”*-egenskaper vars värden är beroende av `Birthdate`.

```
public class Birthday
{
    public DateTime Birthdate { get; set; }
    public string Name { get; set; }
    public int Age
    {
        get
        {
            return this.NextBirthdayDate.Year - this.Birthdate.Year;
        }
    }
    public int DaysUntilNextBirthday
    {
        get
        {
            return (this.NextBirthdayDate - DateTime.Today).Days;
        }
    }
    public DateTime NextBirthdayDate
    {
        get
        {
            var nextBirthday = new DateTime(DateTime.Today.Year,
                this.Birthdate.Month, this.Birthdate.Day);
            if (nextBirthday < DateTime.Today)
            {
                nextBirthday = nextBirthday.AddYears(1);
            }
            return nextBirthday;
        }
    }
}
```

`Birthday` används till att bestämma antalet dagar det är till en persons nästa födelsedag.

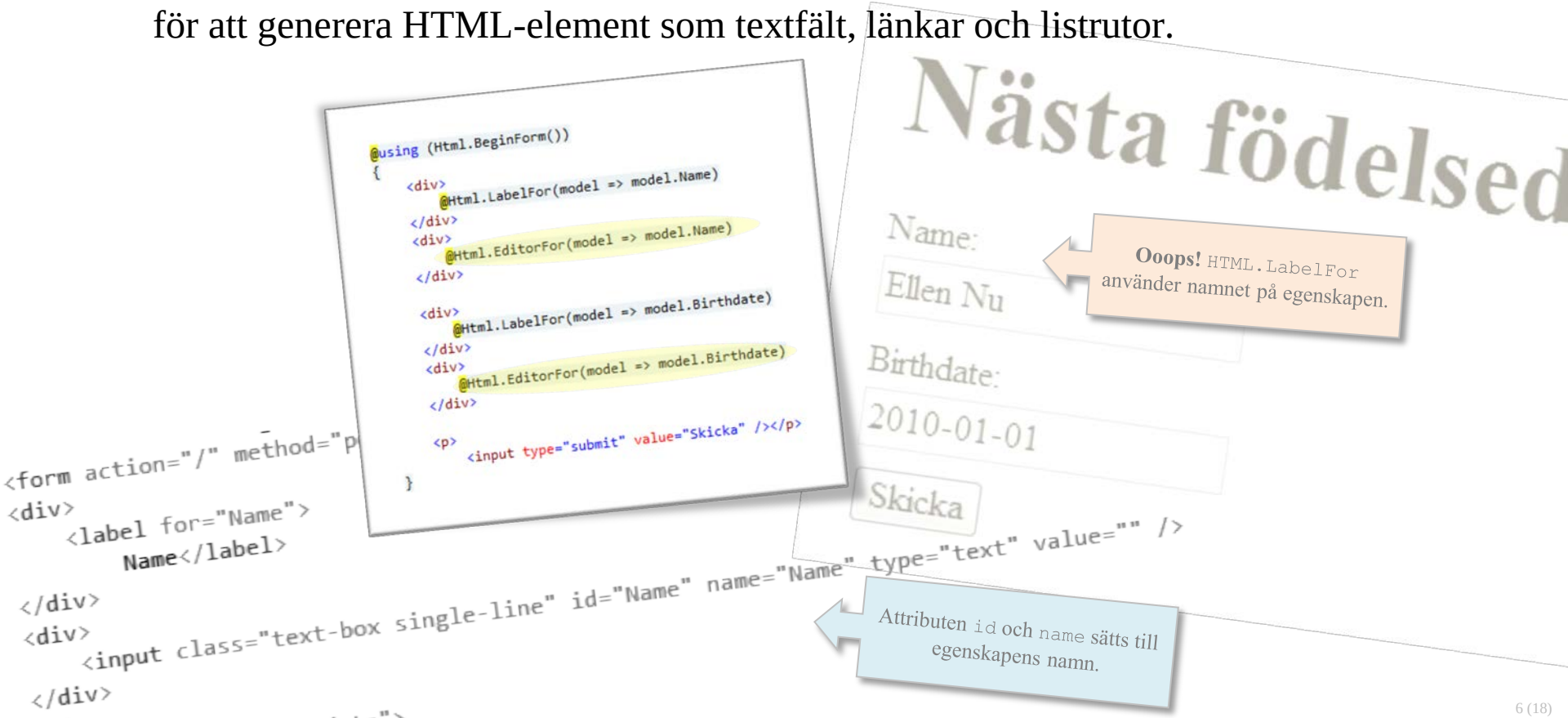
# Ett enkelt formulär för data till modellen…

- …men det finns enklare och bättre sätt.



# Ett enklare formulär med *HTML Helpers*

- ✓ Med hjälp av *HTML Helpers* kan ett formulär genereras.
- ✓ En *HTML Helper* är typiskt en metod som returnerar en sträng och kan användas för att generera HTML-element som textfält, länkar och listrutor.



# Metadata bestämmer vad `Html.LabelFor` renderar

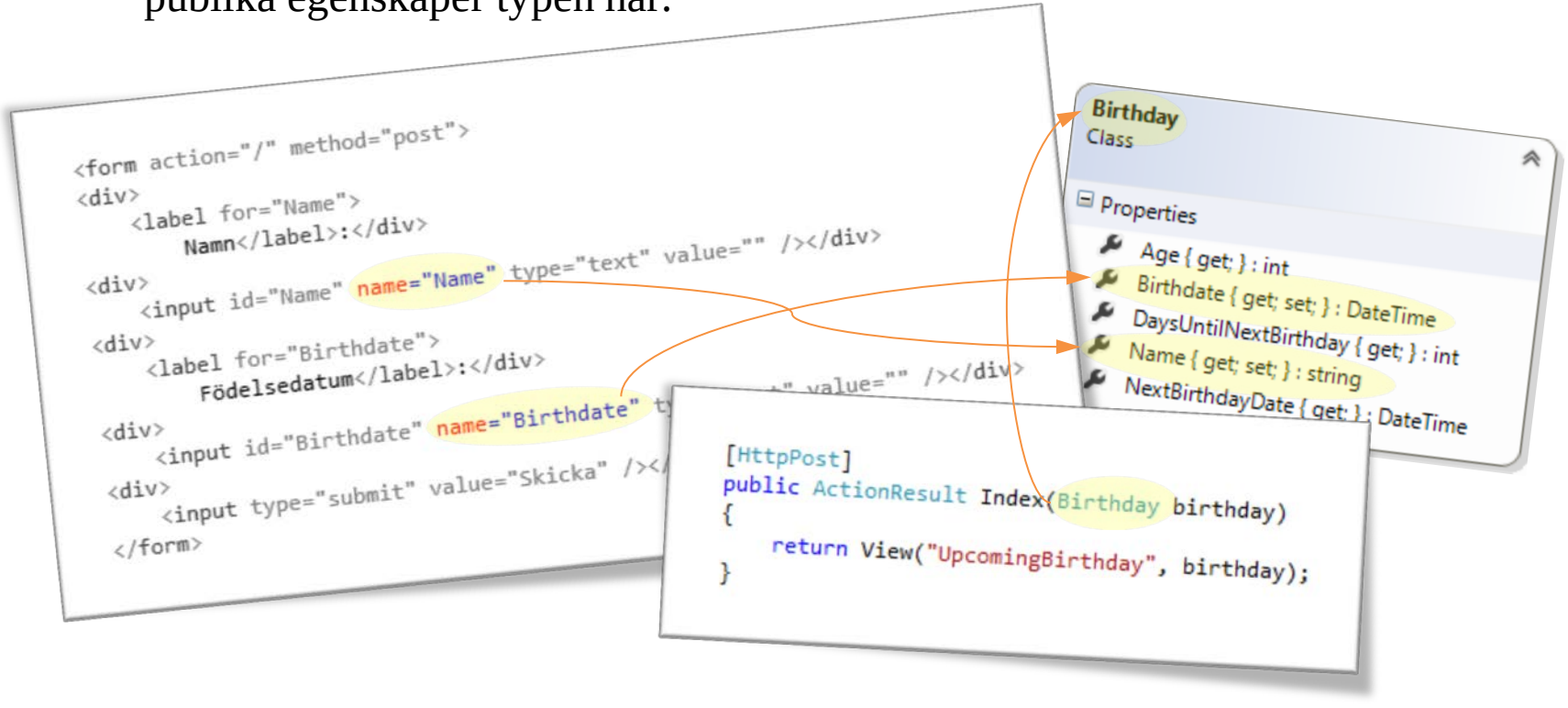
- Genom att dekorera modellklassens egenskaper med metadataattributet `DispalyName` kan du bestämma vad `Html.LabelFor` ska rendera.

# Binda formulärdatat till parametrar

- ✓ Det går att hämta ut formulärdatat manuellt och tilldela modellens egenskaper värdena, men det är smidigare att använda bindningsmekanismen som finns.
- ✓ Formulärdata kan automatiskt bindas till parametrar i en metod.

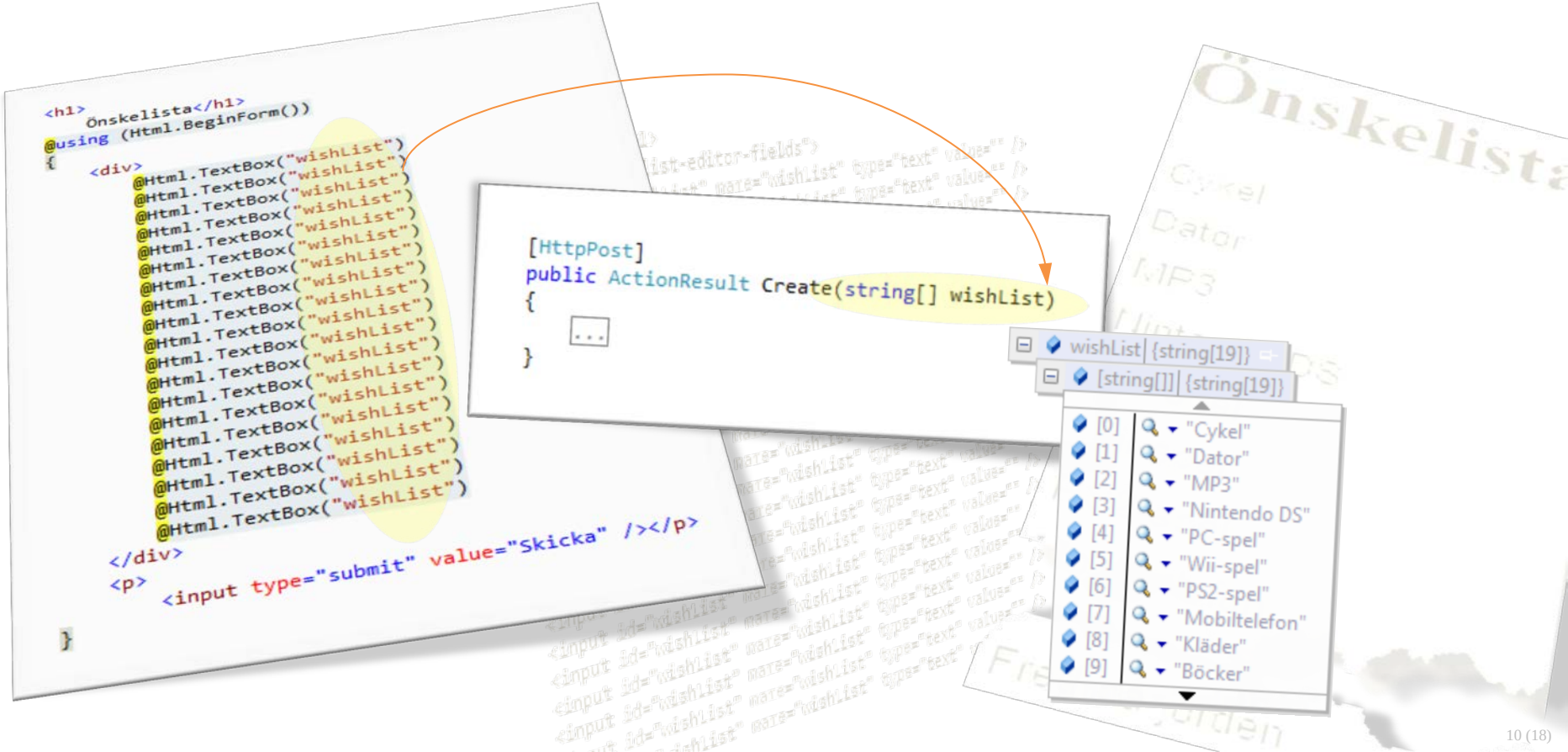
# Automatiskt binda formulärdata till en egen typ

- Formulärdatat kan bindas till egna typer genom att `DefaultModelBinder`, komponenten som ansvarar för konverteringen av formulärdatat, undersöker vilka publika egenskaper typen har.



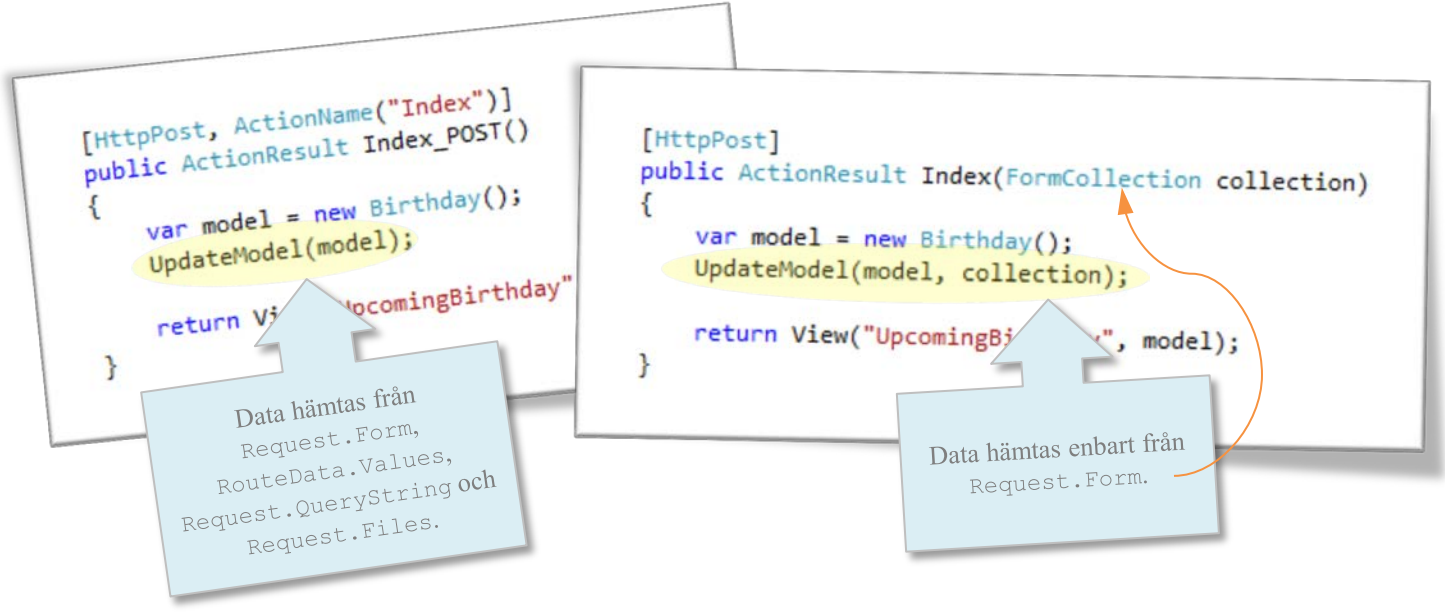
# Binda data som har samma namn

- HTML-element med samma namn i ett formulär kan bindas till arrayer, samlingar eller associativa arrayer.



# Manuellt binda formulärdatat till en egen typ

- Då du behöver större kontroll över hur modellobjektet instansieras kan du manuellt binda formulärdatat till modellobjektet med metoderna `UpdateModel` och `TryUpdateModel`.
  - `UpdateModel` kastar ett undantag om bindningen misslyckas.
  - `TryUpdateModel` returnerar `false` om bindningen misslyckas.



# Modellens status

- MVC-ramverket använder egenskapen `ModelState`, en associativ array, för att lagra information om modellen. I `ModelState` lagras information som modellens status, bindningsfel, inkommande värden, ….

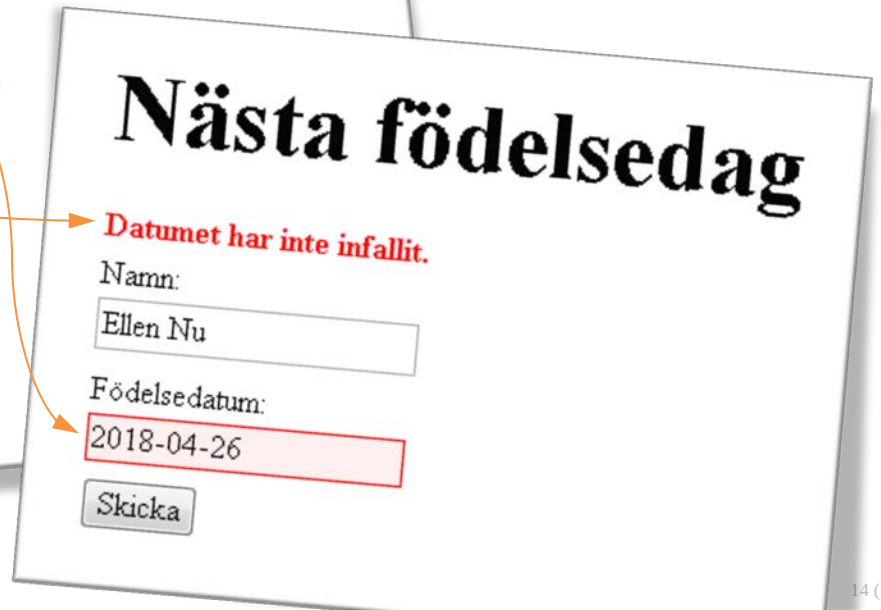
# Felmeddelanden visas med *HTML Helper*

- ✓ Med hjälp av `Html.ValidationSummary` kan eventuella fel i modellen presenteras i ett div-element innehållande ett ul-element.
- ✓ Formulärfält som orsakar fel dekoreras av MVC-ramverket med CSS-klassen `input-validation-error` oavsett om `Html.ValidationSummary` används eller inte.

# Lägga till egna felmeddelanden till `ModelState`

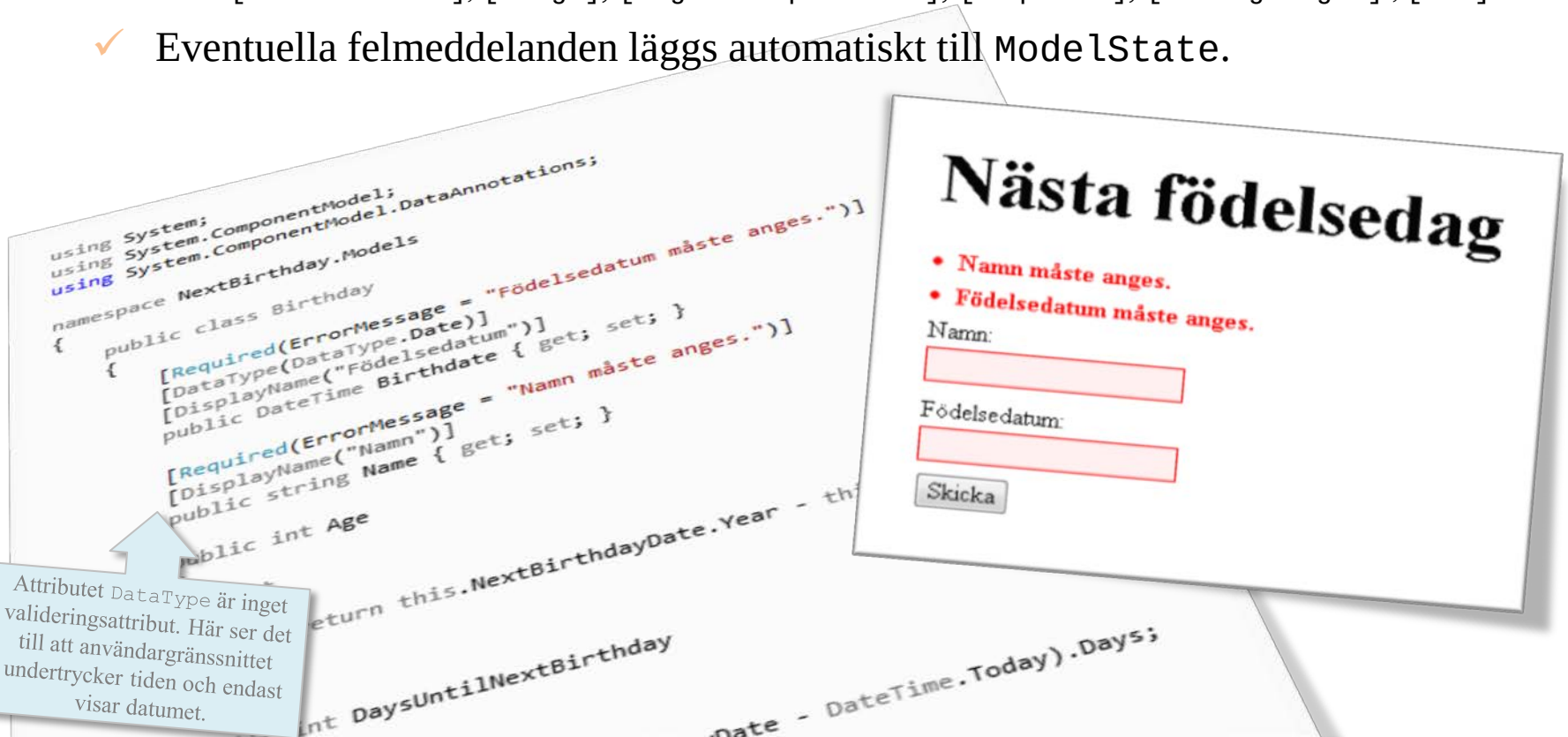
- ✓ Egna felmeddelanden läggs till `ModelState` med metoden `AddModelError`.
  - Om första parametern är namnet på ett formulärfält kopplas felmeddelandet till formulärfältet.

```
[HttpPost]
public ActionResult Index(FormCollection collection)
{
    var model = new Birthday();
    if (TryUpdateModel(model, new[] { "name", "birthdate" }, collection) &&
        model.Birthdate > DateTime.Today)
    {
        ModelState.AddModelError("Birthdate",
            "Datumet har inte infallit.");
    }
    if (ModelState.IsValid)
    {
        return View("UpcomingBirthday", model);
    }
    return View("Index", model);
}
```



# Validering med *data annotation*

- ✓ Lämpligast sätt att validera är att använda *data annotation*. ASP.NET MVC känns vid sex attribut som kan användas av modellklassen, och det går att skapa egna.
  - `[EmailAddress]`, `[Range]`, `[RegularExpression]`, `[Required]`, `[StringLength]` , `[Url]`
- ✓ Eventuella felmeddelanden läggs automatiskt till `ModelState`.



# Placering av felmeddelanden

- Genom att använda `Html.ValidationMessageFor` kan felmeddelanden placeras var som helst i formuläret.

Värdet `true` ser till att fel knutna till ett fält inte visas i en osorterad lista.

```
@using (Html.
{
    @Html.ValidationSummary(true, "Fel inträffade. Korrigera det som är fel och försök ig
    <div>
        @Html.LabelFor(model => model.Name):
    </div>
    <div>
        @Html.EditorFor(model => model.Name)
        @Html.ValidationMessageFor(model => model.Name)
    </div>

    <div>
        @Html.LabelFor(model => model.Birthdate)
    </div>
    <div>
        @Html.EditorFor(model => model.Birthdate)
        @Html.ValidationMessageFor(model => model.Birthda
    </div>

    <p>
        <input type="submit" value="Skicka" /></p>

}
```

# Validering på klienten

- ASP.NET MVC 4 använder sig av *data annotation* ihop med jQuery och jQuery Validation för validering på klienten (”*Unobtrusive Client Validation*”).
- Valideringsmeddelanden visas och döljs dynamiskt och formuläret postas inte om fälten inte klarar valideringen på klienten.

# Mer information

- ✓ I kurslitteraturen kapitel 20-23 hittar du mer information om *”model templates”*, bindning, validering och *”unobtrusive Ajax*”.
- ✓ På http://www.asp.net/mvc/tutorials/mvc-4/getting-started-with-aspnet-mvc4/adding-validation-to-the-model finns exempel på validering. På http://www.asp.net/mvc/tutorials/older-versions/models-(data)/performing-simple-validation-cs finns flera ”gamla” exempel på hur validering kan implementeras.