Using Partially Applied Functions in Modeling

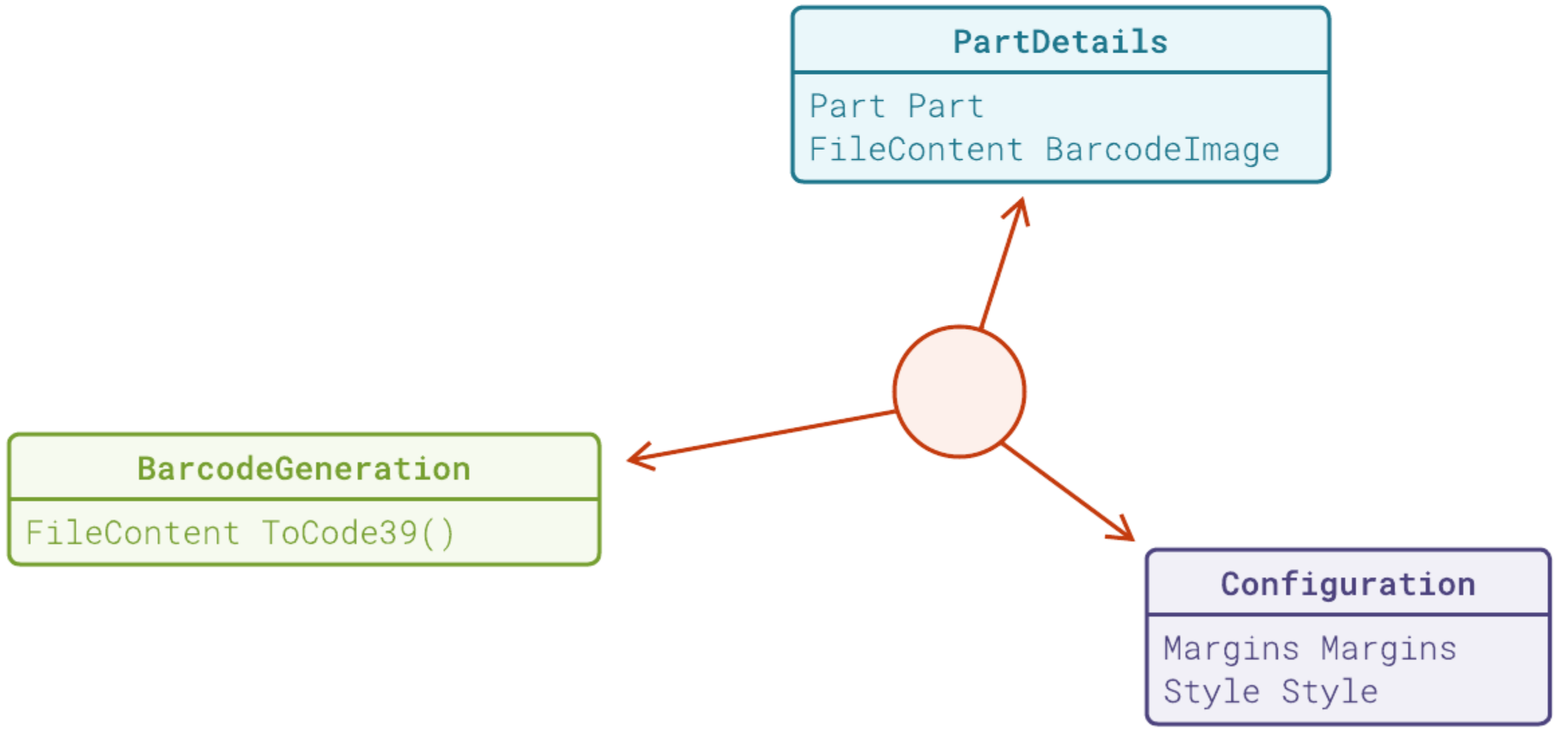
Zoran Horvat

CEO at Coding Helmet

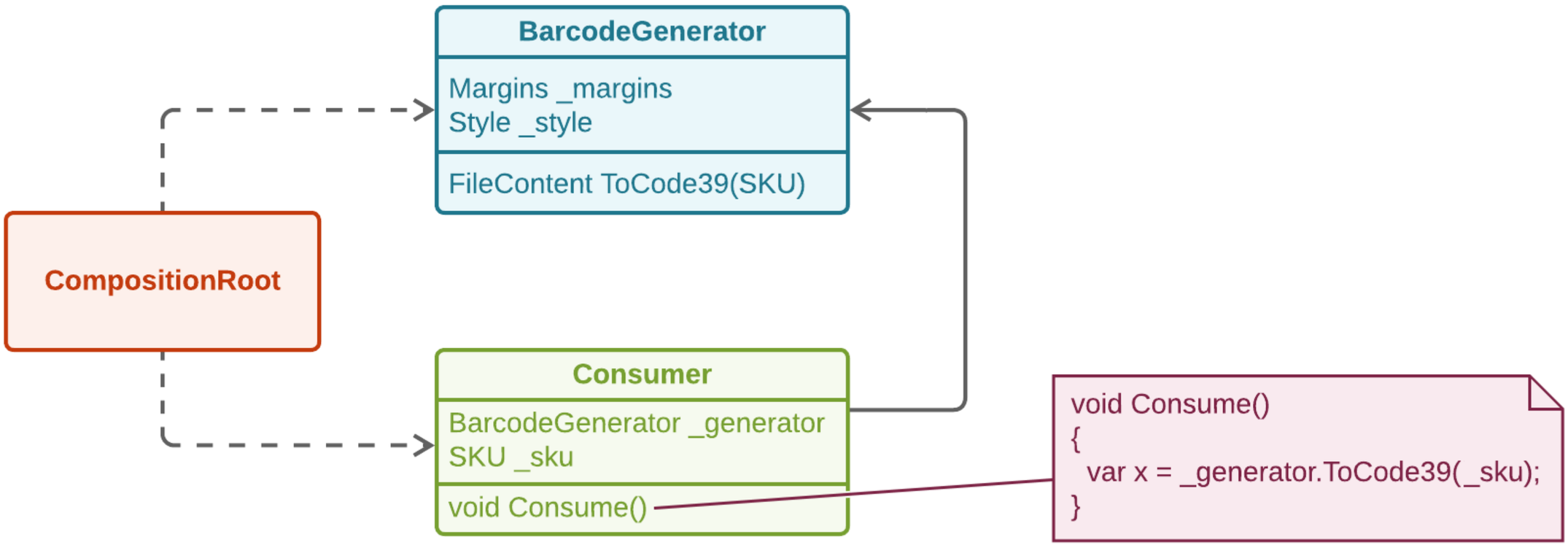
@zoranh75 https://codinghelmet.com

```
17
18      public Part Part { get; set; } = null!;
19      public FileContent BarcodeImage { get; set; } = null!;
20
21      public void OnGet(Guid id)
22      {
23          this.Part = this.Parts.Find(id);
24          this.BarcodeImage = this.Part.Sku.ToCode39(this.Margins, this.Style);
25      }
26
27      private BarcodeGeneration.Margins Margins => new(
28          Horizontal: 20, Vertical: 10, BarHeight: 200);
29
30      private BarcodeGeneration.Style Style => new(
31          ThinBarWidth: 4, ThickBarWidth: 10, GapWidth: 6, Padding: 6, Antialias: false);
32  }
33
```

```
17
18      public Part Part { get; set; } = null!;
19      public FileContent BarcodeImage { get; set; } = null!;
20
21      public void OnGet(Guid id)
22      {
23          this.Part = this.Parts.Find(id);
24          this.BarcodeImage = this.Part.Sku.ToCode39(                    );
25      }
26
27
28
29
30
31
32  }
33
```

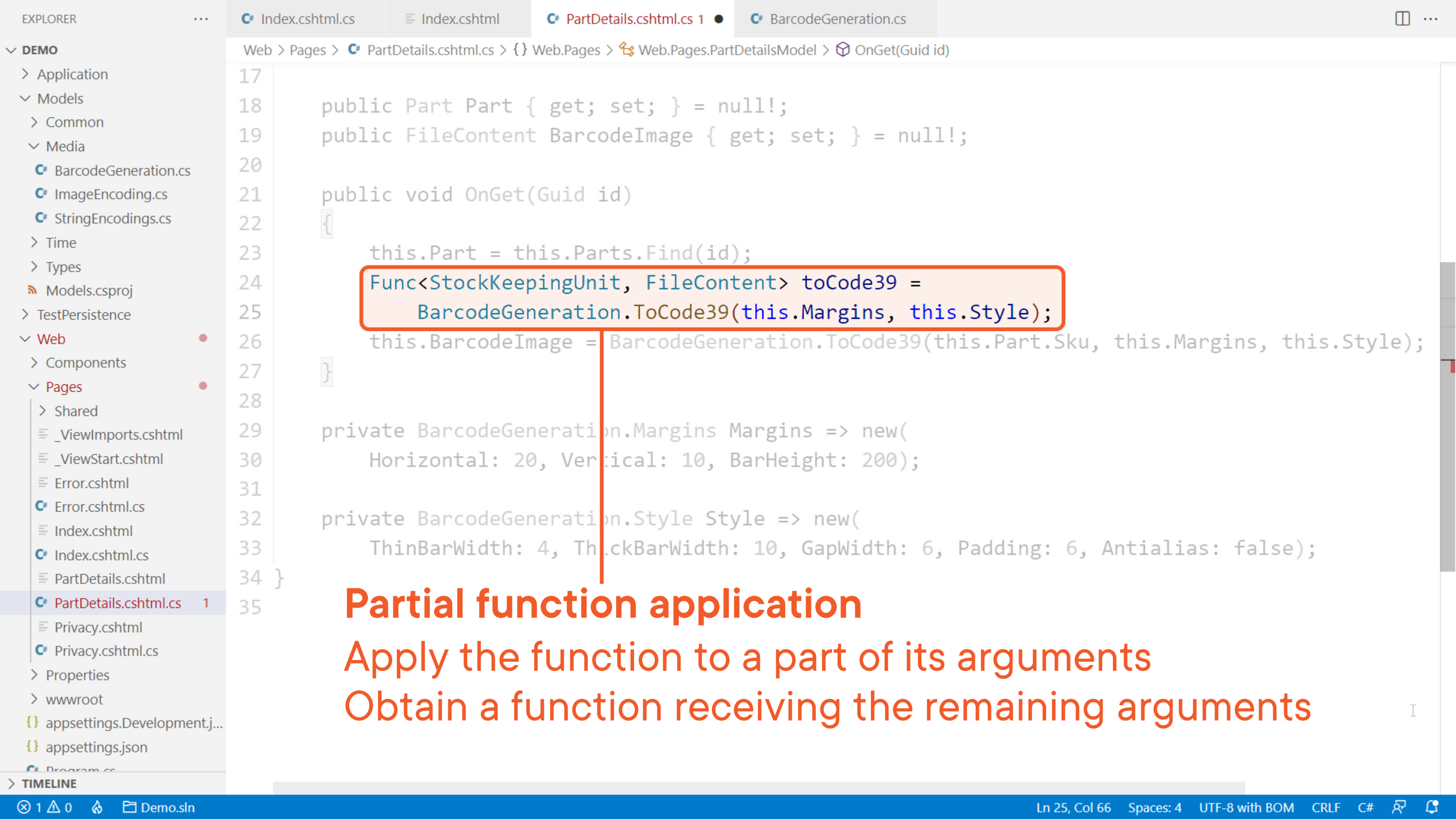


Object-oriented Composition



```
17
18      public Part Part { get; set; } = null!;
19      public FileContent BarcodeImage { get; set; } = null!;
20
21      public void OnGet(Guid id)
22      {
23          this.Part = this.Parts.Find(id);
24          Func<StockKeepingUnit, FileContent> toCode39 =
25              BarcodeGeneration.ToCode39(this.Margins, this.Style);
26          this.BarcodeImage = BarcodeGeneration.ToCode39(this.Part.Sku, this.Margins, this.Style);
27      }
28
29      private BarcodeGeneration.Margins Margins => new(
30          Horizontal: 20, Vertical: 10, BarHeight: 200);
31
32      private BarcodeGeneration.Style Style => new(
33          ThinBarWidth: 4, ThickBarWidth: 10, GapWidth: 6, Padding: 6, Antialias: false);
34  }
35
```

```
17
18      public Part Part { get; set; } = null!;
19      public FileContent BarcodeImage { get; set; } = null!;
20
21      public void OnGet(Guid id)
22      {
23          this.Part = this.Parts.Find(id);
24          Func<StockKeepingUnit, FileContent> toCode39 =
25              BarcodeGeneration.ToCode39(this.Margins, this.Style);
26          this.BarcodeImage = BarcodeGeneration.ToCode39(this.Part.Sku, this.Margins, this.Style);
27      }
28
29      private BarcodeGeneration.Margins Margins => new(
30          Horizontal: 20, Vertical: 10, BarHeight: 200);
31
32      private BarcodeGeneration.Style Style => new(
33          ThinBarWidth: 4, ThickBarWidth: 10, GapWidth: 6, Padding: 6, Antialias: false);
34  }
35
```


EXPLORER
Index.cshtml.cs
Index.cshtml
PartDetails.cshtml.cs 1
BarcodeGeneration.cs
Web > Pages > PartDetails.cshtml.cs > { } Web.Pages > Web.Pages.PartDetailsModel > OnGet(Guid id)
DEMO
Application
Models
Common
Media
BarcodeGeneration.cs
ImageEncoding.cs
StringEncodings.cs
Time
Types
Models.csproj
TestPersistence
Web
Components
Pages
Shared
_ViewImports.cshtml
_ViewStart.cshtml
Error.cshtml
Error.cshtml.cs
Index.cshtml
Index.cshtml.cs
PartDetails.cshtml
PartDetails.cshtml.cs 1
Privacy.cshtml
Privacy.cshtml.cs
Properties
wwwroot
appsettings.Development.j...
appsettings.json
TIMELINE
17
18 public Part Part { get; set; } = null!;
19 public FileContent BarcodeImage { get; set; } = null!;
20
21 public void OnGet(Guid id)
22 {
23 this.Part = this.Parts.Find(id);
24 Func<StockKeepingUnit, FileContent> toCode39 =
25 BarcodeGeneration.ToCode39(this.Margins, this.Style);
26 this.BarcodeImage = BarcodeGeneration.ToCode39(this.Part.Sku, this.Margins, this.Style);
27 }
28
29 private BarcodeGeneration.Margins Margins => new(
30 Horizontal: 20, Vertical: 10, BarHeight: 200);
31
32 private BarcodeGeneration.Style Style => new(
33 ThinBarWidth: 4, ThickBarWidth: 10, GapWidth: 6, Padding: 6, Antialias: false);
34 }
35
Partial function application
Apply the function to a part of its arguments
Obtain a function receiving the remaining arguments
1 0
Demo.sln
Ln 25, Col 66
Spaces: 4
UTF-8 with BOM
CRLF
C#

```
class SomeClass
{
  public SomeClass(
    Func<int, int> dependency) =>
    Dependency = dependency;

  private Func<int, int> Dependency
    { get; }
}
```

- **Inject a** Func/Action **delegate like a dependency**

```
class SomeClass
{
  public SomeClass(
    Func<int, int> dependency) =>
    Dependency = dependency;

  private Func<int, int> Dependency
    { get; }
}
```

- **Inject a** Func/Action **delegate like a dependency**

- **Delegate is an object, like any other**

```
class SomeClass
{
  public SomeClass(
    Func<int, int> dependency) =>
    Dependency = dependency;

  private Func<int, int> Dependency
    { get; }
}
```

- **Inject a** Func/Action **delegate like a dependency**

- **Delegate is an object, like any other**

  It may include captured variables (the closure)

```
class SomeClass
{
  public SomeClass(
    Func<int, int> dependency) =>
    Dependency = dependency;

  private Func<int, int> Dependency
    { get; }

  private void Demo()
  {
    var y = this.Dependency.Invoke(5);
  }
}
```

- **Inject a** Func/Action **delegate like a dependency**

- **Delegate is an object, like any other**

  It may include captured variables (the closure)

- **Call the delegate's** Invoke **method to execute it**

```
class SomeClass
{
  public SomeClass(
    Func<int, int> dependency) =>
    Dependency = dependency;

  private Func<int, int> Dependency
    { get; }

  private void Demo()
  {
    var y = this.Dependency.Invoke(5);
    var z = this.Dependency(5);
  }
}
```

- **Inject a** Func/Action **delegate like a dependency**

- **Delegate is an object, like any other**

  It may include captured variables (the closure)

- **Call the delegate's** Invoke **method to execute it**
- **Or use shorthand syntax for the call**

```
class SomeClass
{
  public SomeClass(
    Func<int, int> dependency) =>
    Dependency = dependency;

  private Func<int, int> Dependency
    { get; }

  private void Demo()
  {
    var y = this.Dependency.Invoke(5);
    var z = this.Dependency(5);
  }
}

private Func<int, int, int> f;
```

- **Inject a** Func/Action **delegate like a dependency**

- **Delegate is an object, like any other**

  It may include captured variables (the closure)

- **Call the delegate's** Invoke **method to execute it**
- **Or use shorthand syntax for the call**

- **Arguments have no names**

```
class SomeClass
{
  public SomeClass(
    Func<int, int> dependency) =>
    Dependency = dependency;

  private Func<int, int> Dependency
    { get; }

  private void Demo()
  {
    var y = this.Dependency.Invoke(5);
    var z = this.Dependency(5);
  }
}

private Func<int, int, int> f;

private Func<FileContent,
             StockKeepingUnit> g;
```

- **Inject a** Func/Action **delegate like a dependency**
- **Delegate is an object, like any other**
  - It may include captured variables (the closure)
- **Call the delegate's** Invoke **method to execute it**
- **Or use shorthand syntax for the call**
- **Arguments have no names**
- **Argument types can help avoid mistakes**

```
class SomeClass
{
  public SomeClass(
    Func<int, int> dependency) =>
    Dependency = dependency;

  private Func<int, int> Dependency
    { get; }

  private void Demo()
  {
    var y = this.Dependency.Invoke(5);
    var z = this.Dependency(5);
  }
}

private Func<int, int, int> f;

private Func<FileContent,
             StockKeepingUnit> g;
```

- **Inject a** Func/Action **delegate like a dependency**

- **Delegate is an object, like any other**

  It may include captured variables (the closure)

- **Call the delegate's** Invoke **method to execute it**
- **Or use shorthand syntax for the call**

- **Arguments have no names**

- **Argument types can help avoid mistakes**

  Still possible to inject *any* function with this signature

```
class SomeClass
{
  public SomeClass(
    Func<int, int> dependency) =>
    Dependency = dependency;

  private Func<int, int> Dependency
    { get; }

  private void Demo()
  {
    var y = this.Dependency.Invoke(5);
    var z = this.Dependency(5);
  }
}

private Func<int, int, int> f;

private Func<FileContent,
             StockKeepingUnit> g;
```

- **Inject a** Func/Action **delegate like a dependency**

- **Delegate is an object, like any other**

  It may include captured variables (the closure)

- **Call the delegate's** Invoke **method to execute it**
- **Or use shorthand syntax for the call**

- **Arguments have no names**

- **Argument types can help avoid mistakes**

  Still possible to inject *any* function with this signature

Summary

Partial function application

- Supply a sublist of function's arguments
- Obtain a function receiving the rest

Partial function application in C#

- Transform a static method into
  a delegate with shorter argument list

Using strongly-typed delegates

- Allow extension methods on themselves
- We can transform functions
  into new functions

Summary

Practical use of partial application

- Separated configuration arguments from the function's primary target
- Injected partially-applied function as a dependency

Important takeaway

- Partial function application simplifies code

Up Next:

Substituting Inheritance with Discriminated Unions