Decorator-Pattern mit ObjectPascal

- Fallstudie
  - Lösungsansatz: Klassische Vererbung
  - Lösungsansatz: Objektattribute
  - Lösungsansatz: Decorator
- Definition
- Refaktorisierungs-Beispiel
- Ausblick

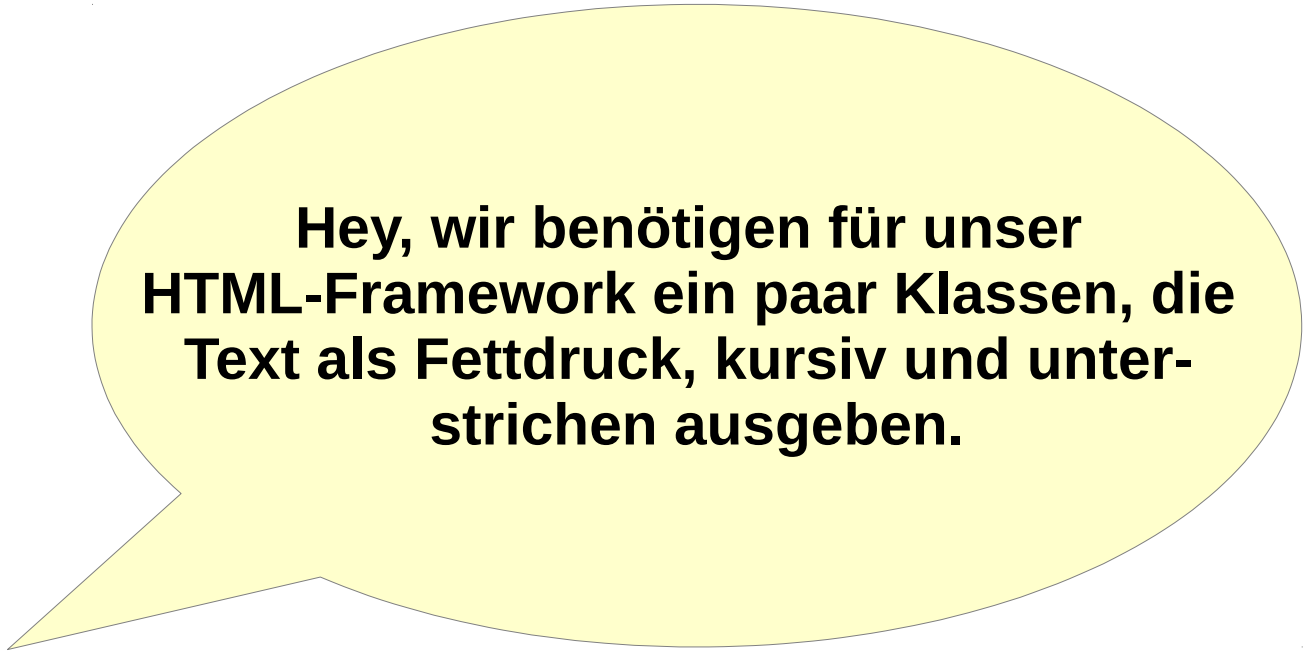
Fallstudie

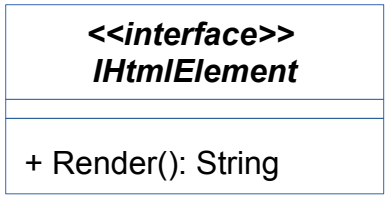
- Lösungsansatz: Klassische Vererbung
- Lösungsansatz: Objektattribute
- Lösungsansatz: Decorator

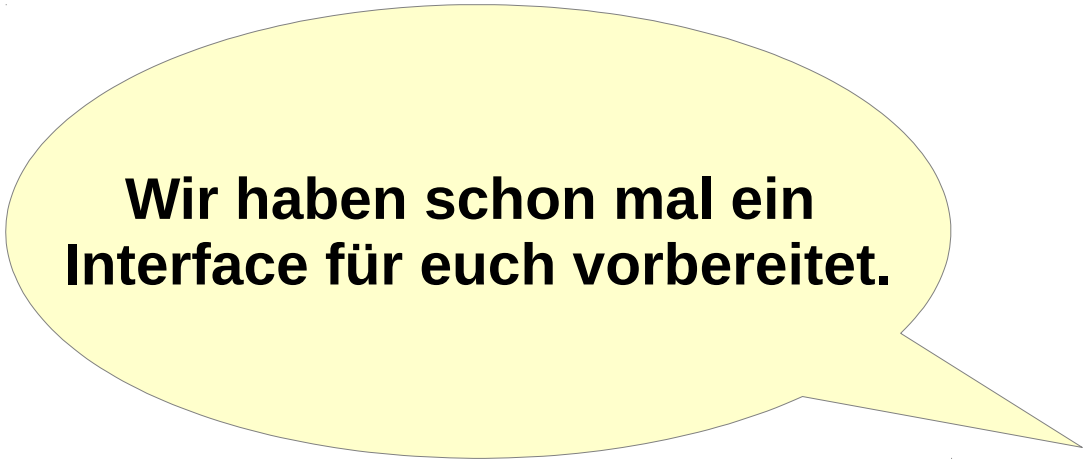
Definition

Refaktorisierungs-Beispiel

Ausblick




<<interface>>
IHtmlElement
+ Render(): String


Wir haben schon mal ein
Interface für euch vorbereitet.

Fallstudie

Lösungsansatz: Klassische Vererbung

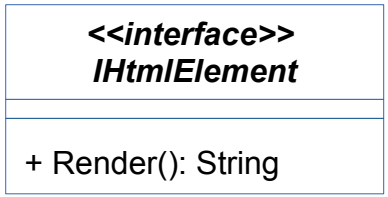
Lösungsansatz: Objektattribute

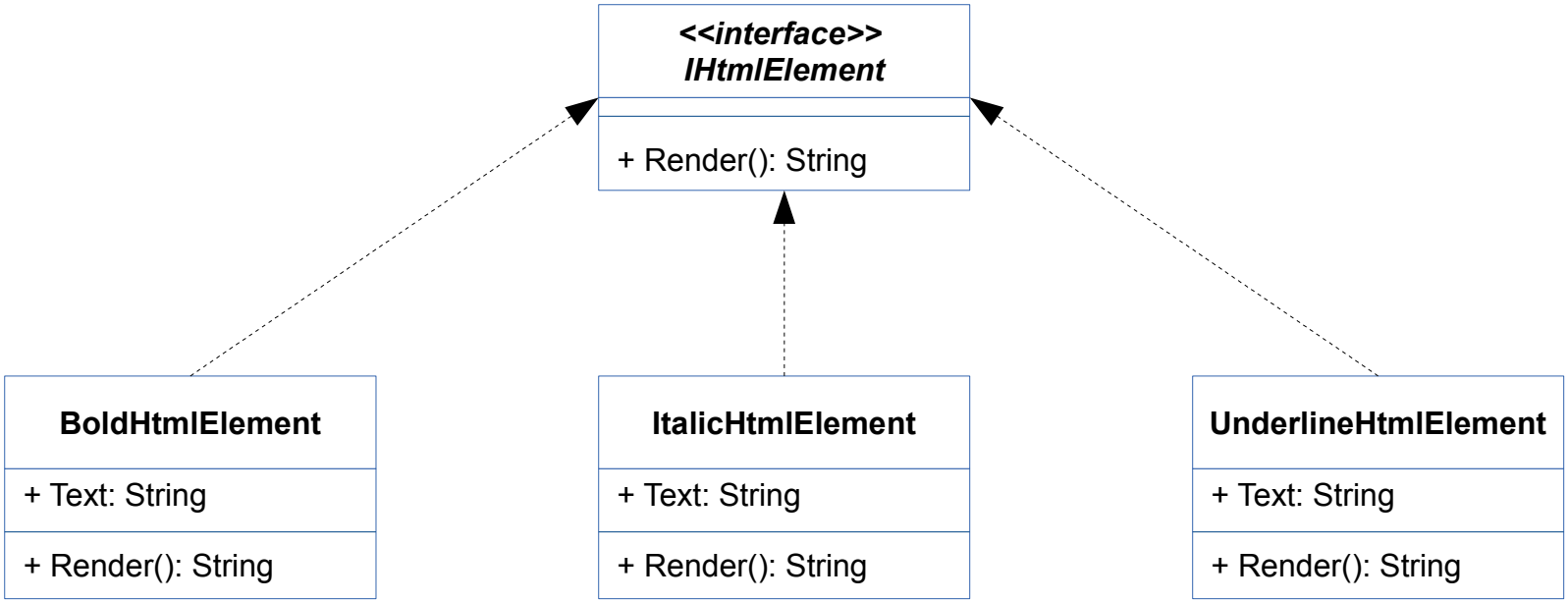
Lösungsansatz: Decorator

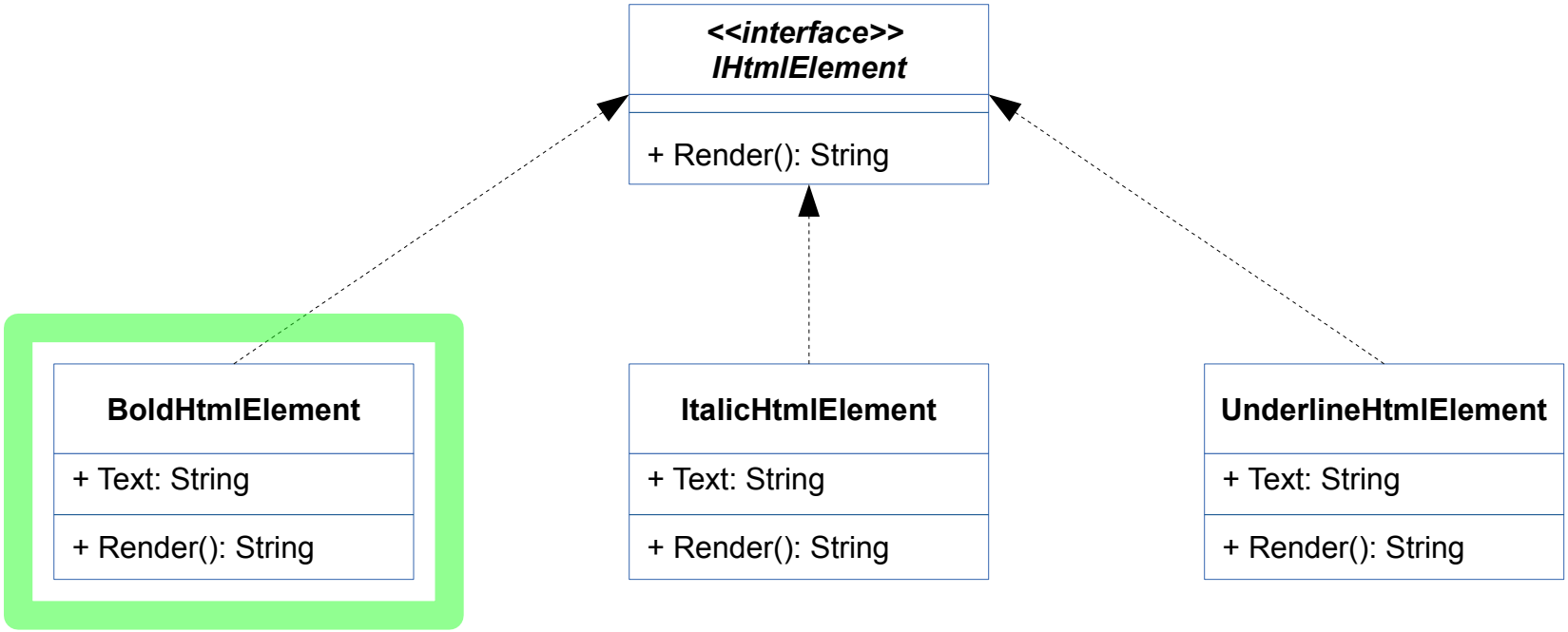
Definition

Refaktorisierungs-Beispiel

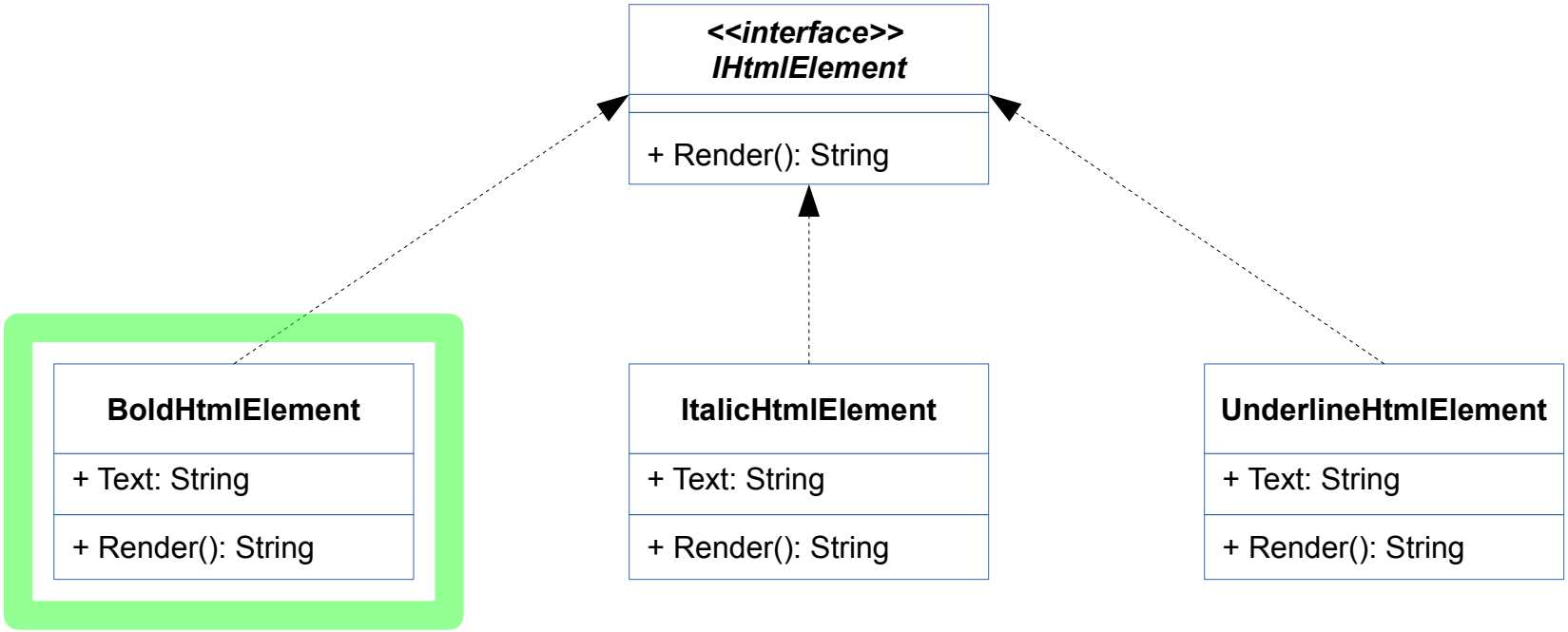
Ausblick


<<interface>>
IHtmlElement
+ Render(): String

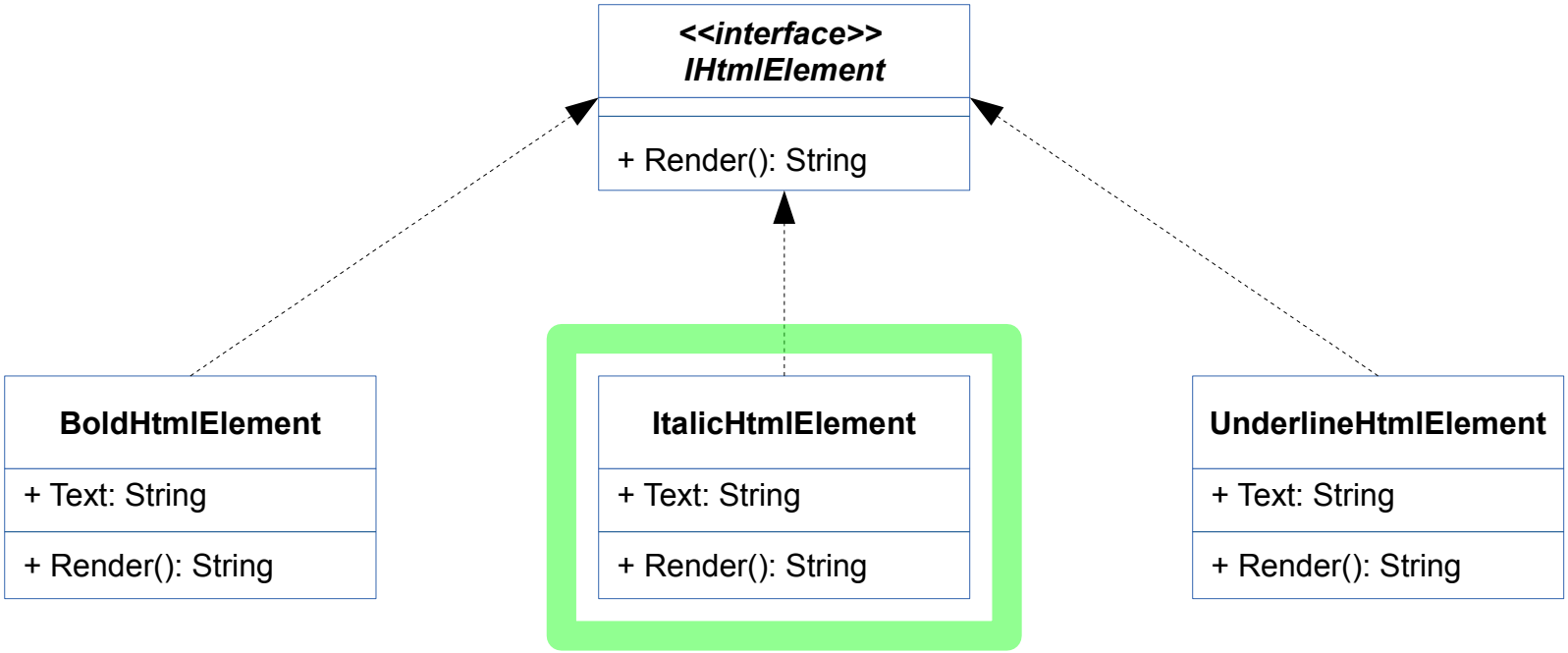




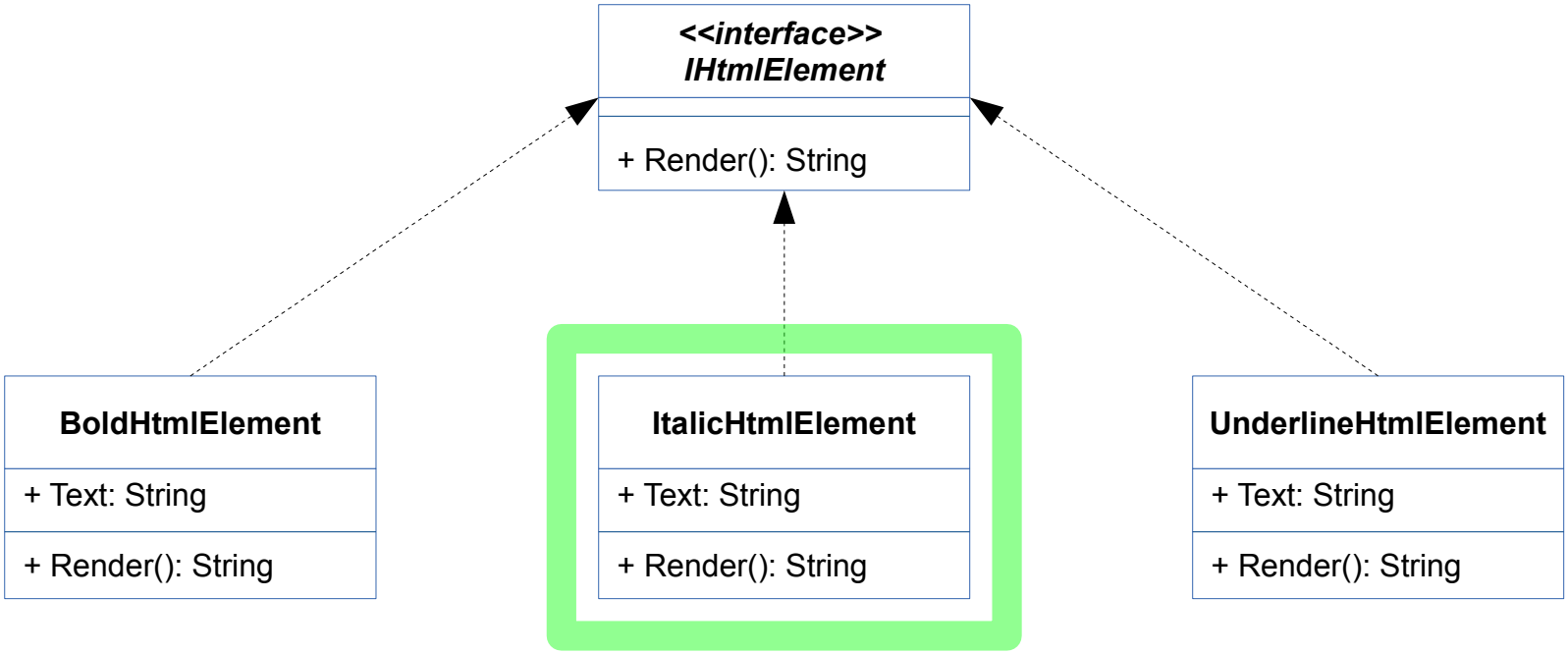

```
TBoldHtmlElement = class(IHtmlElement)
  public
    Text: String;
  public
    function Render: String;
end;
```



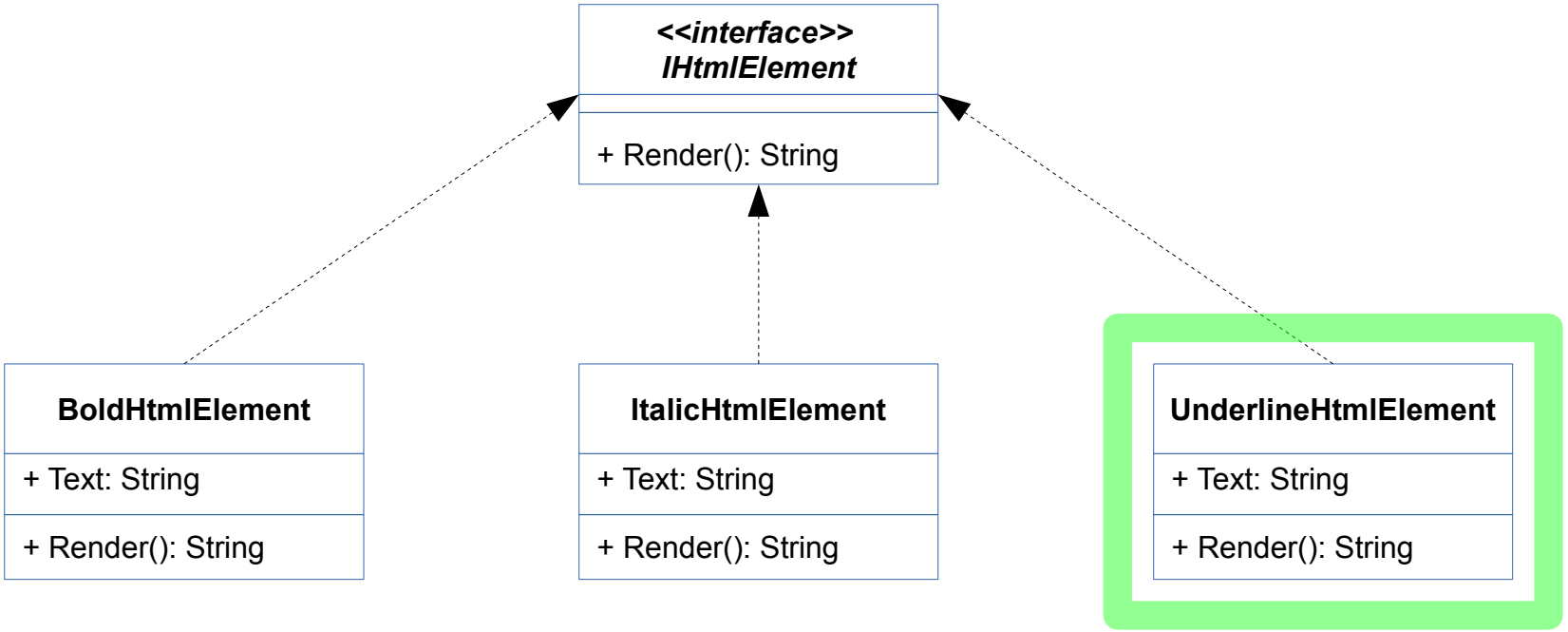

```
function TBoldHtmlElement.Render: String;
begin
  Result := '<b>' + Self.Text + '</b>';
end;
```



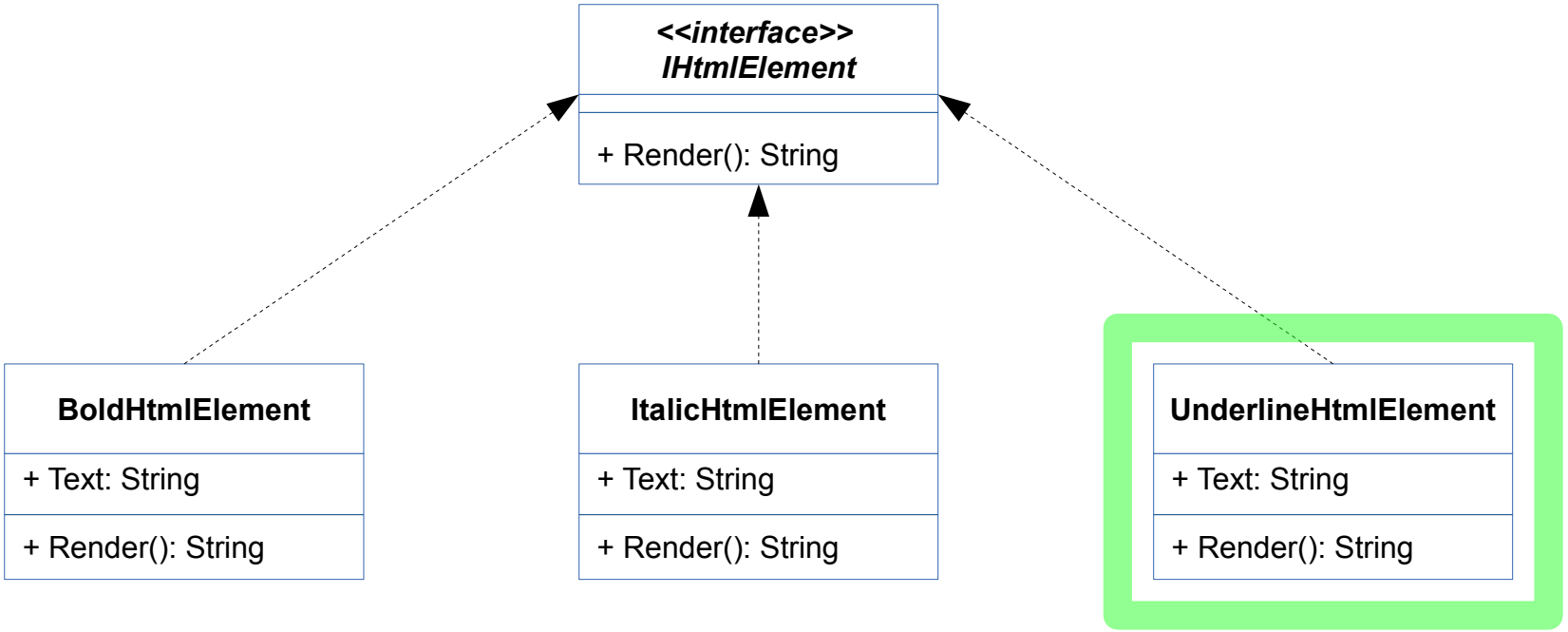

```
TItalicHtmlElement = class(IHtmlElement)
  public
    Text: String;
  public
    function Render: String;
end;
```



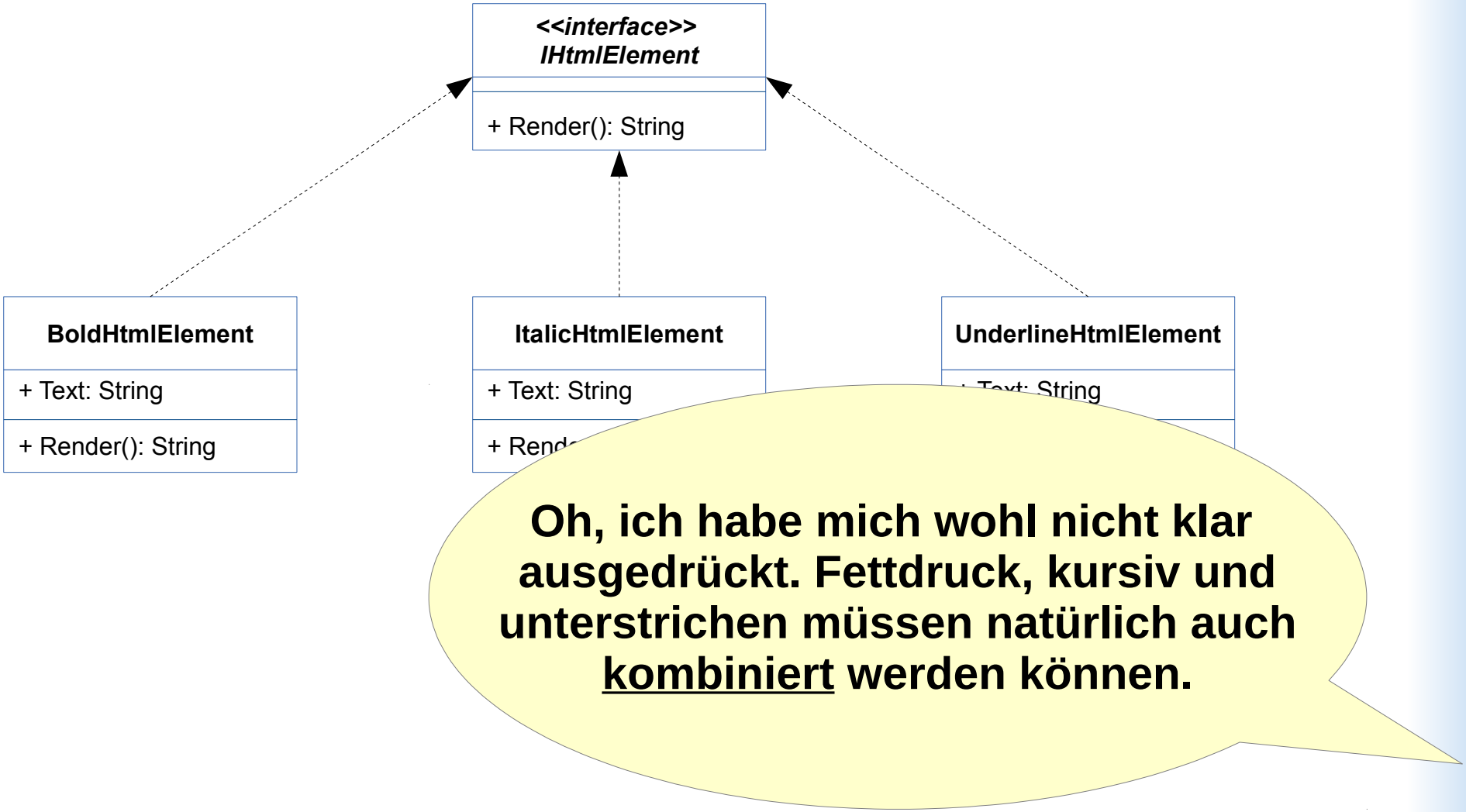

```
function TItalicHtmlElement.Render: String;
begin
  Result := '<i>' + Self.Text + '</i>';
end;
```

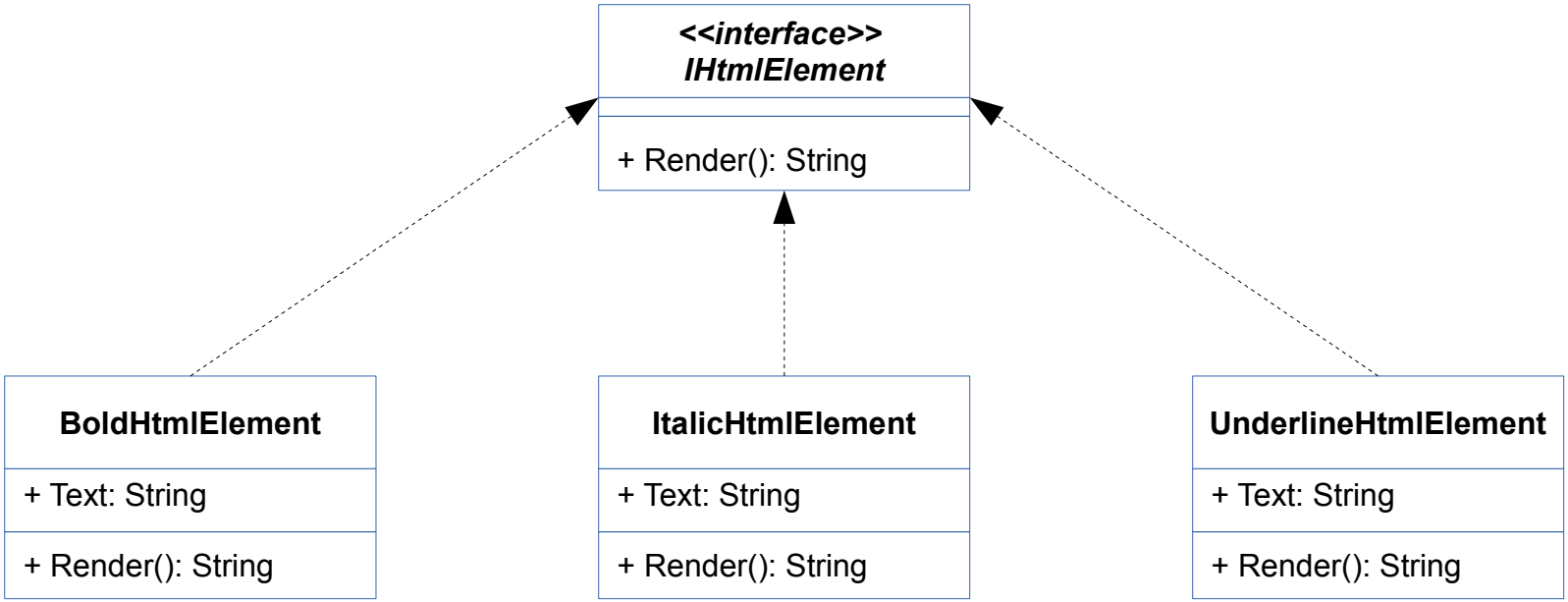


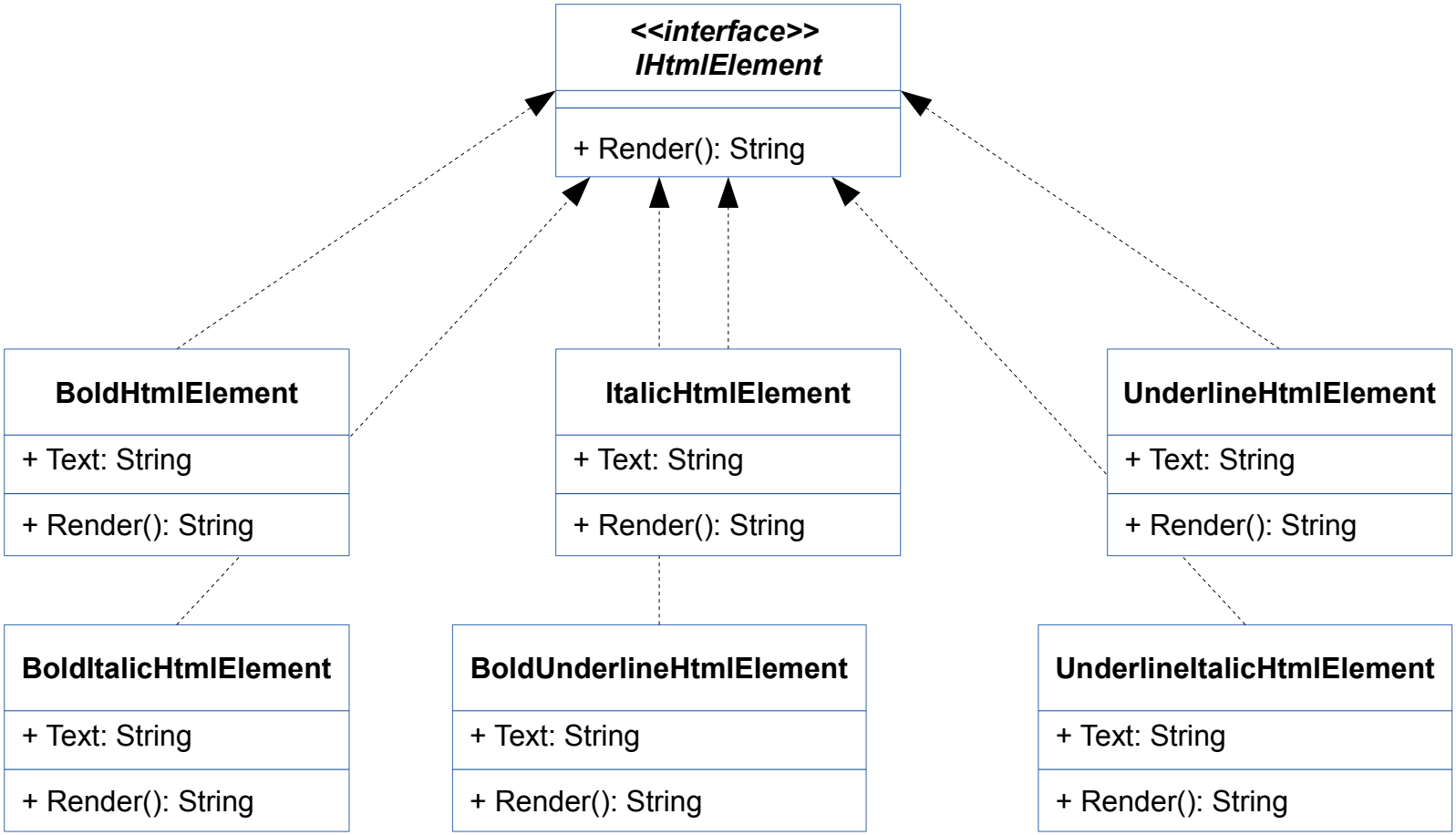

```
TUnderlineHtmlElement = class(IHtmlElement)
  public
    Text: String;
  public
    function Render: String;
end;
```

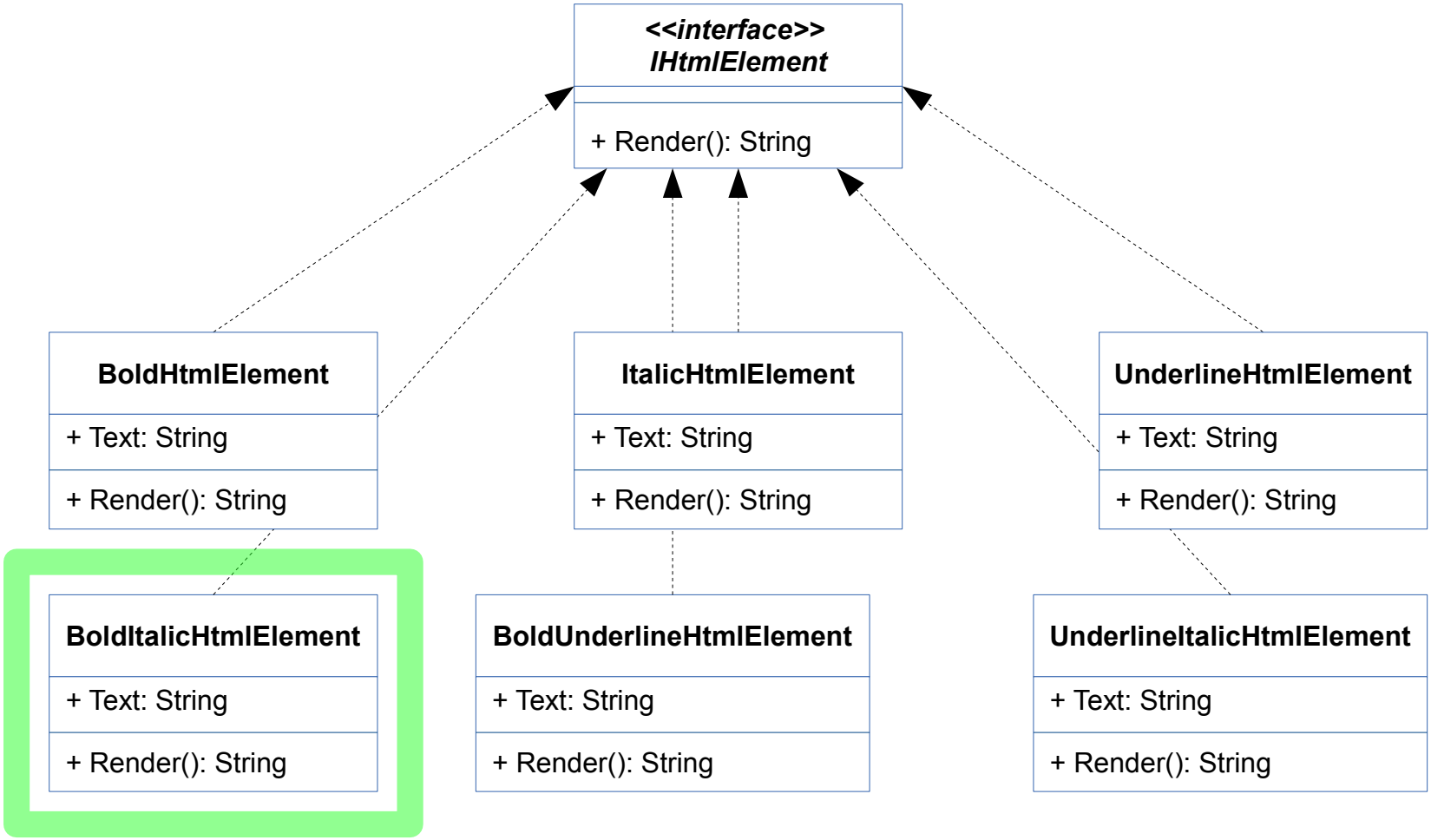



```
function TUnderlineHtmlElement.Render: String;
begin
  Result := '<u>' + Self.Text + '</u>';
end;
```

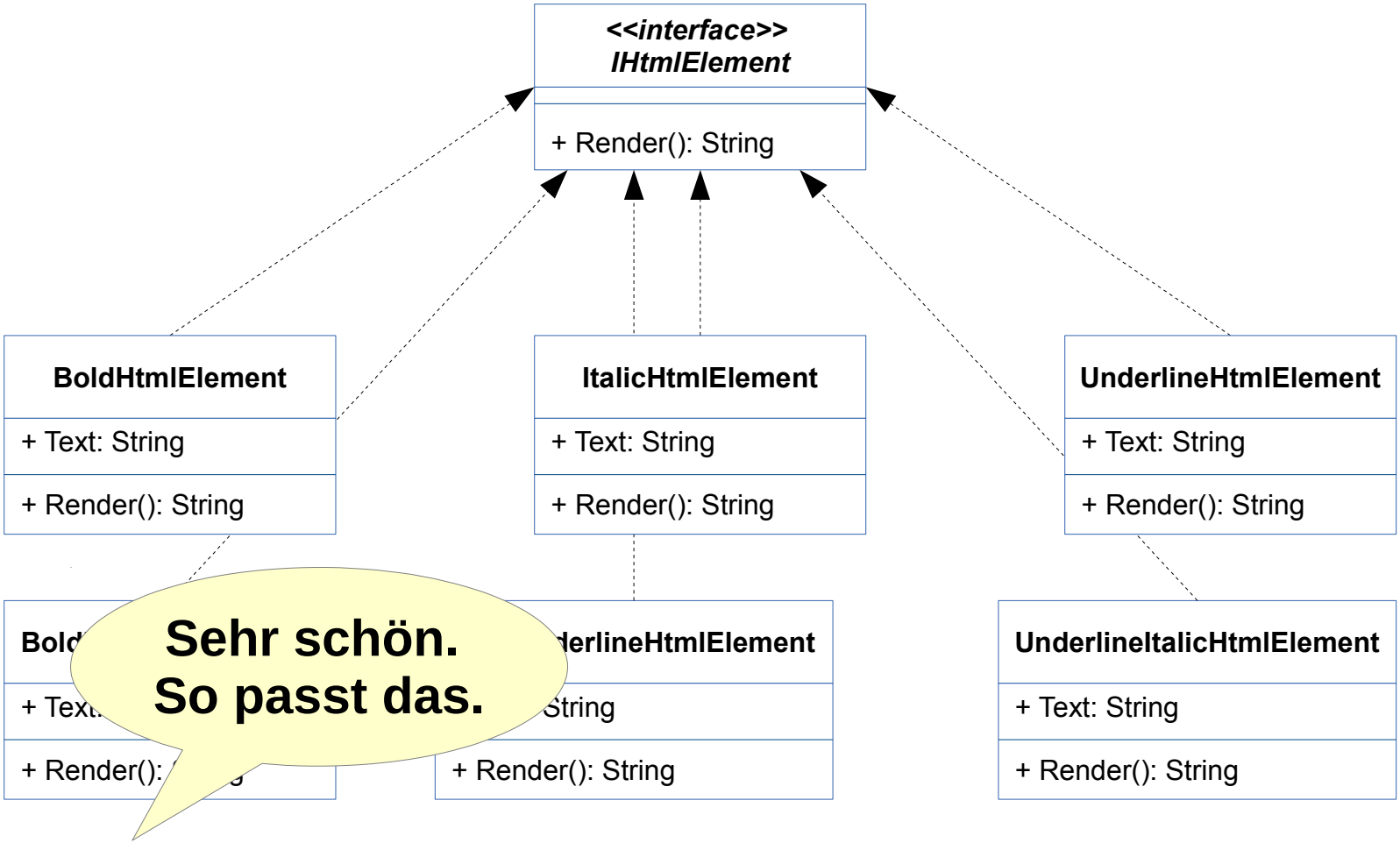


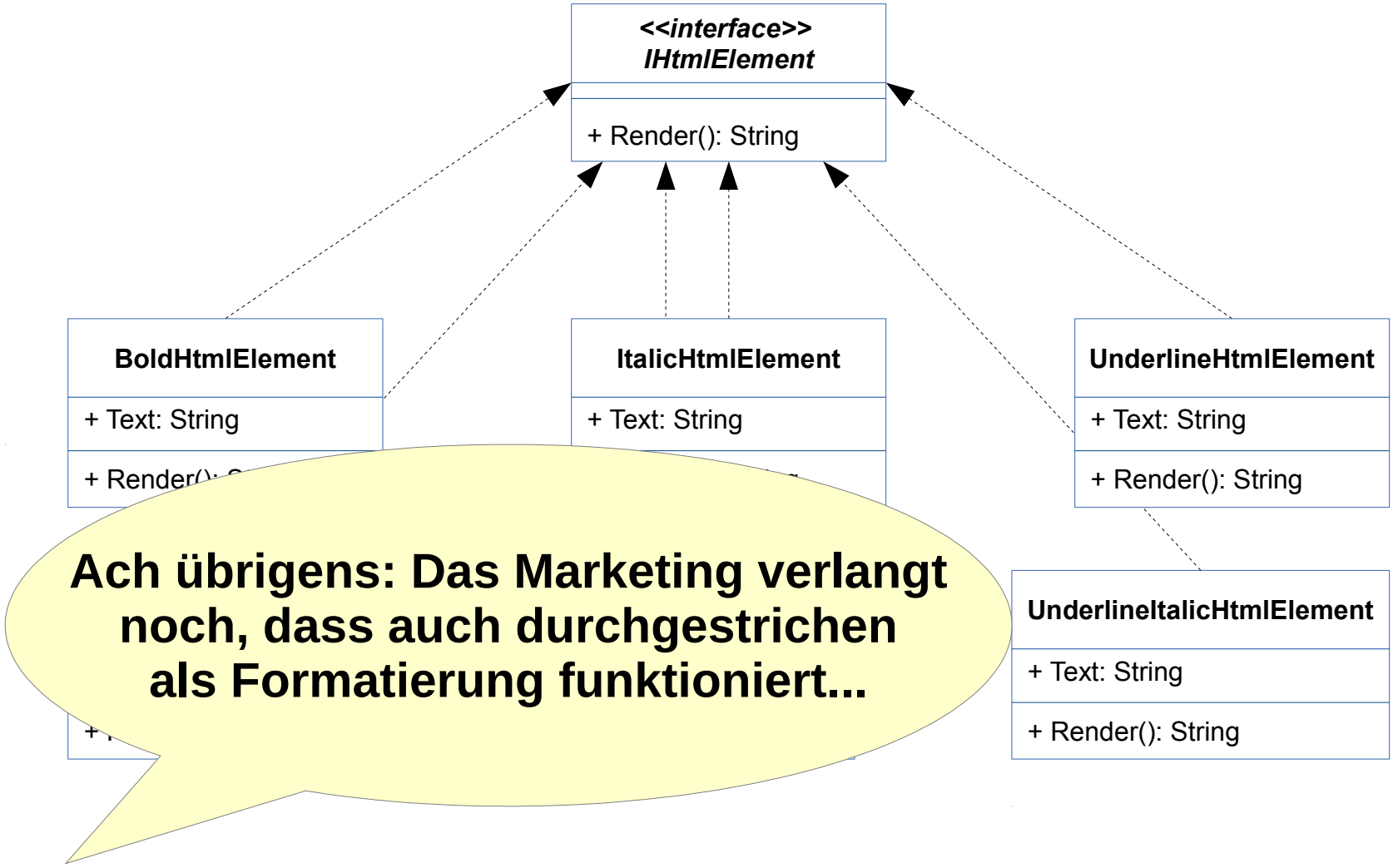


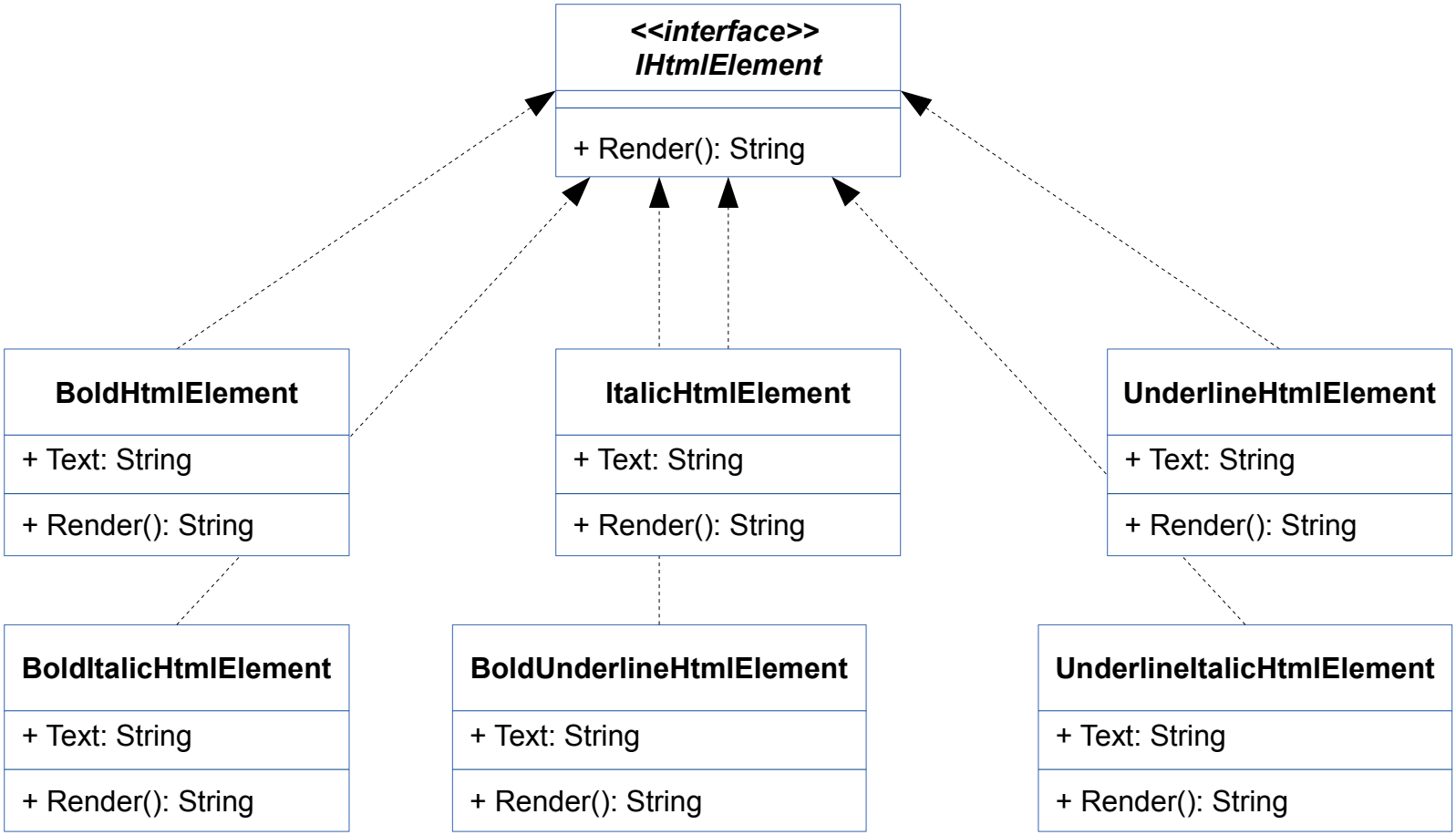


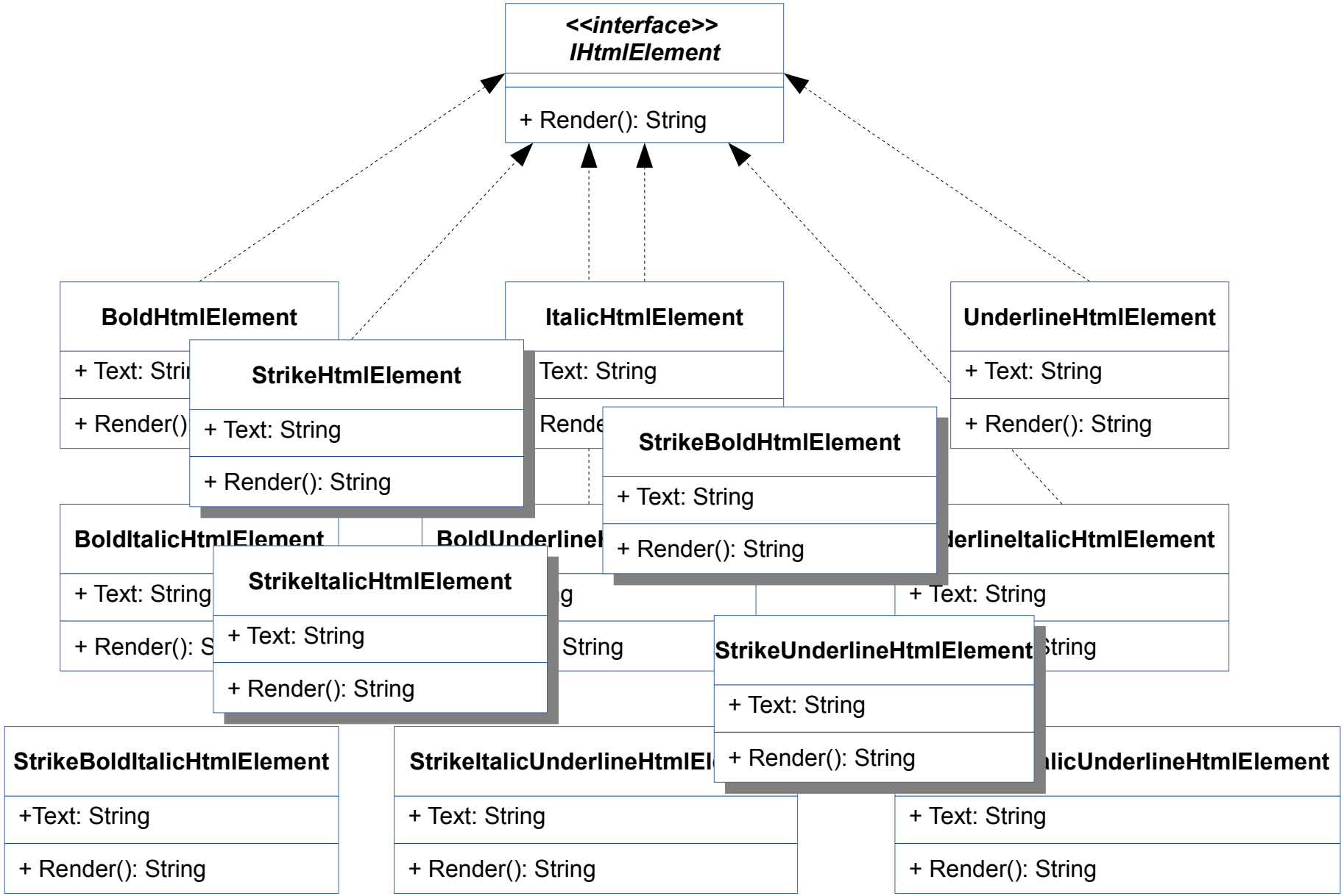


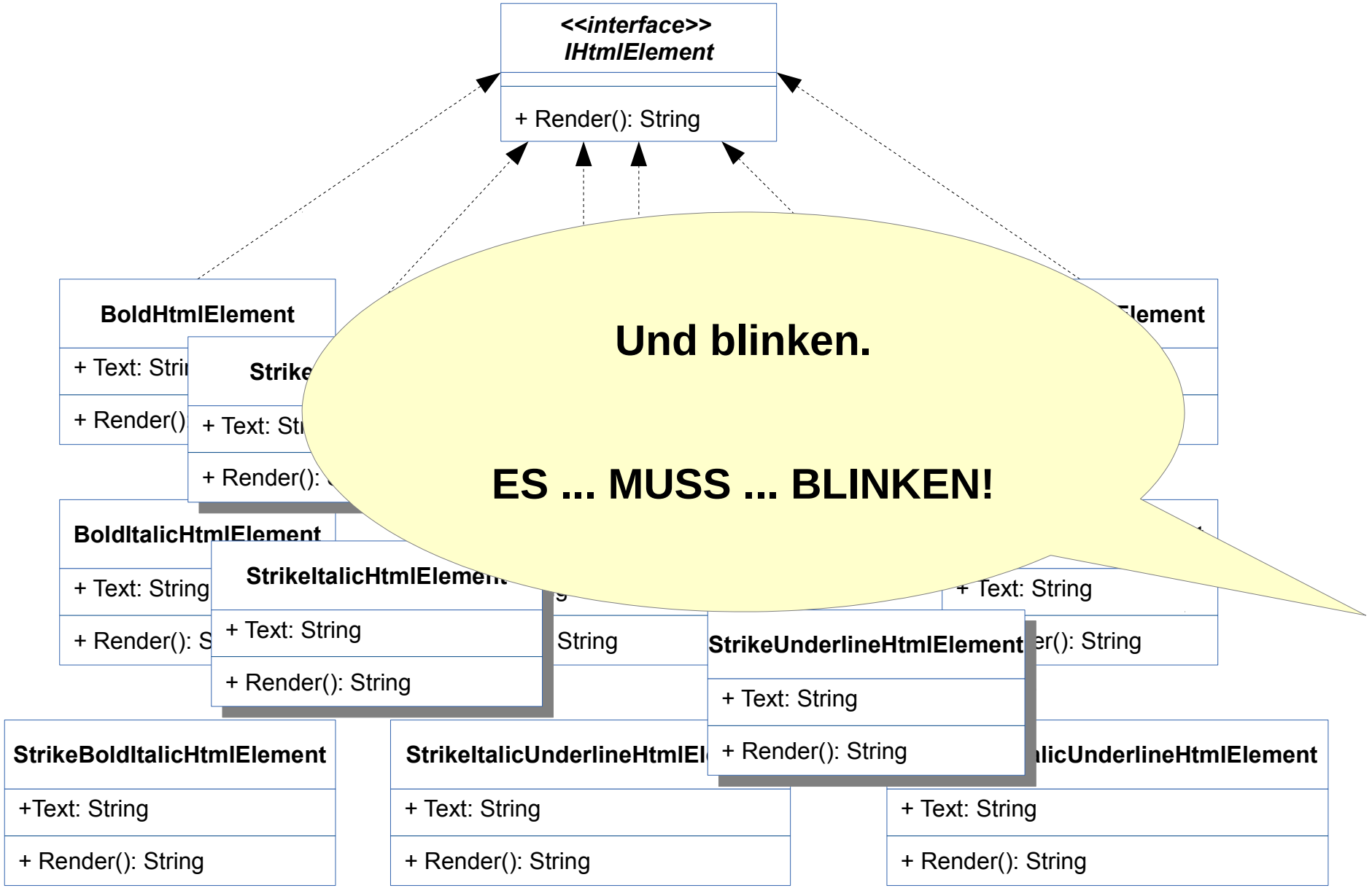

```
function TBoldItalicHtmlElement.Render: String;
begin
  Result := '<b><i>' + Self.Text + '</i></b>';
end;
```

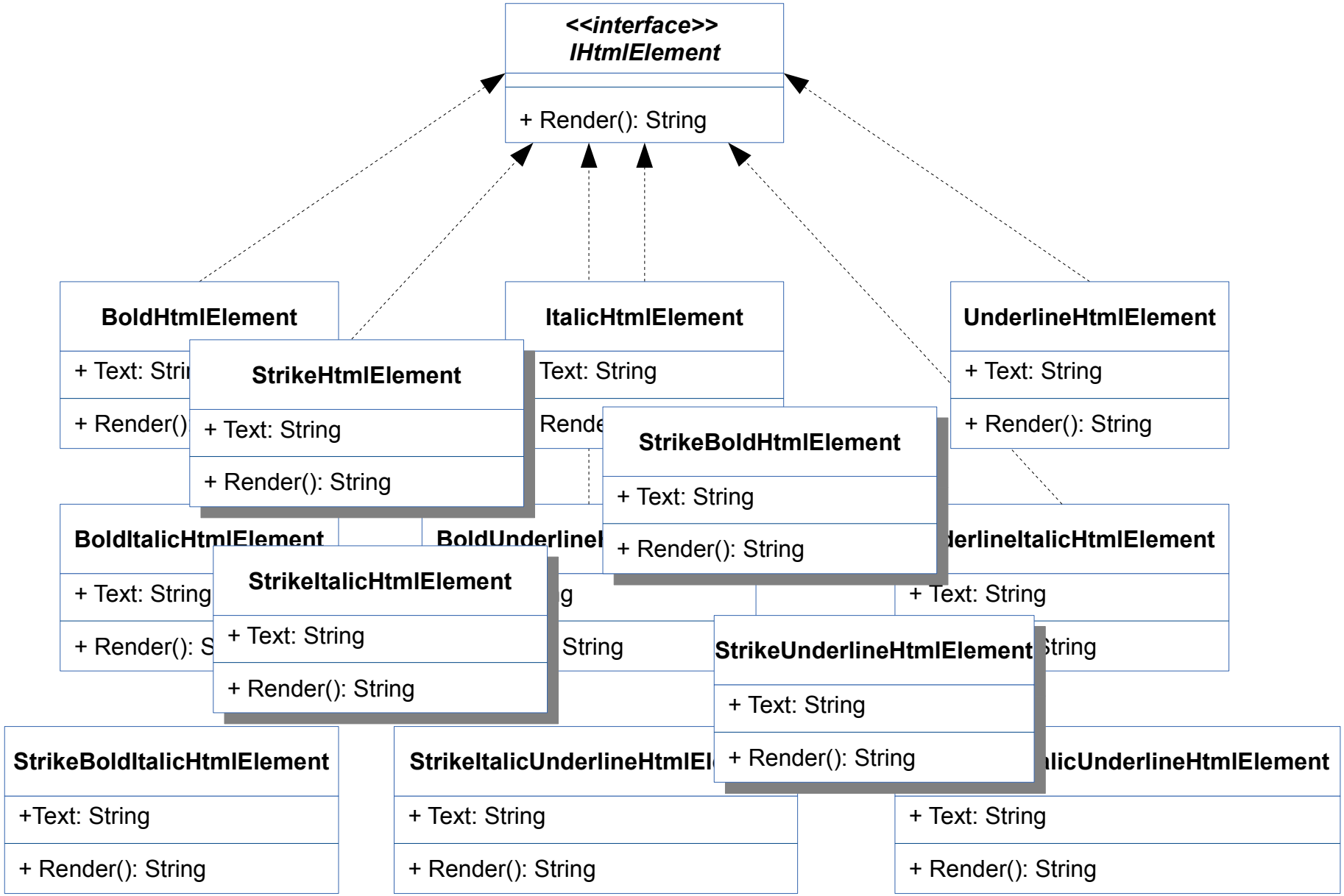


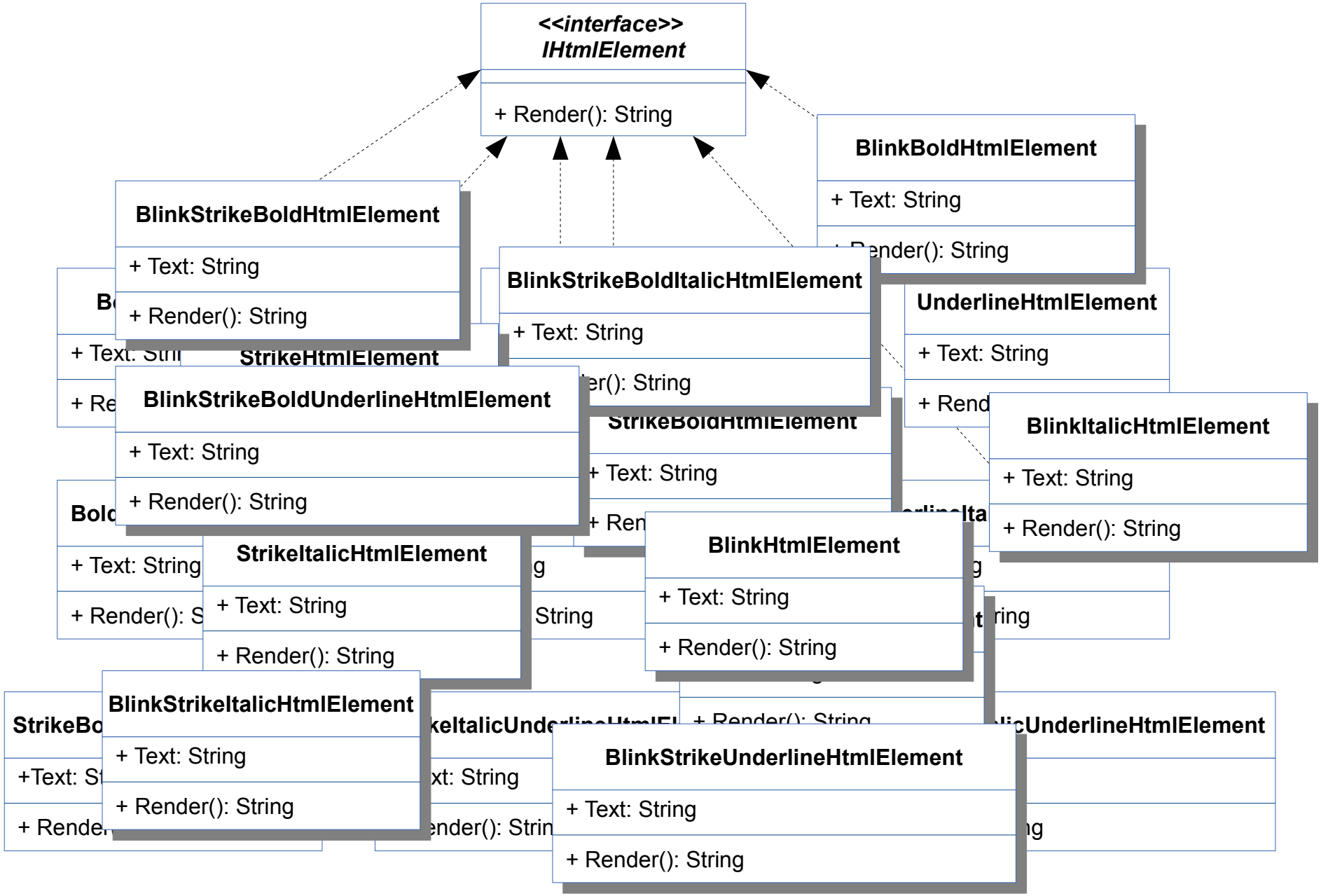


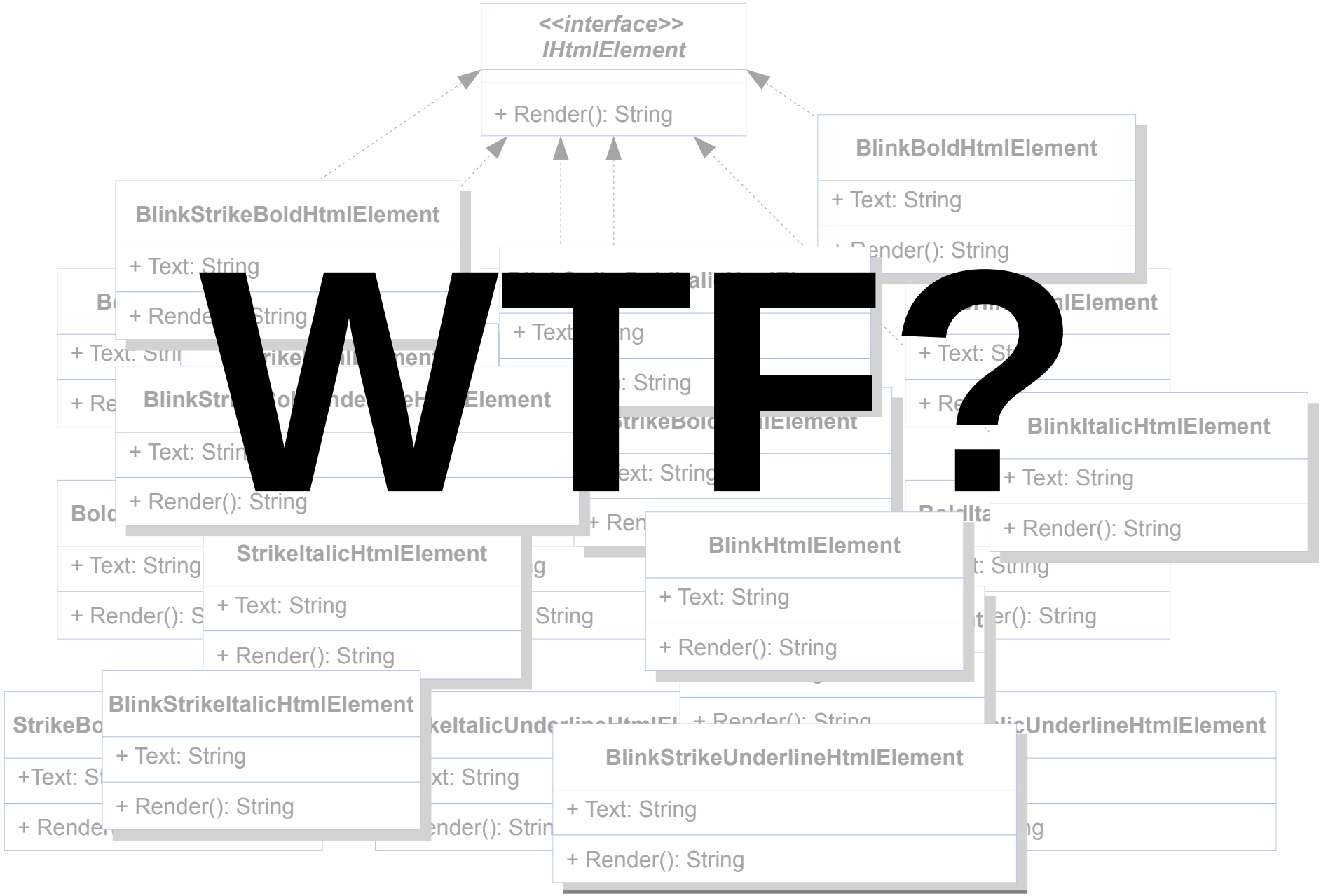












Fallstudie

Lösungsansatz: Klassische Vererbung

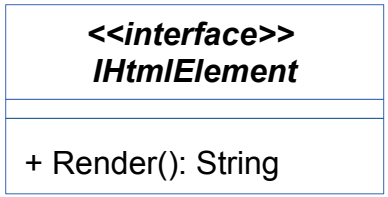
Lösungsansatz: Objektattribute

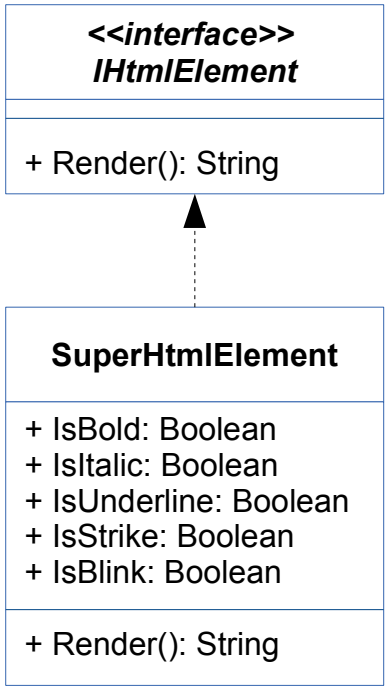
Lösungsansatz: Decorator

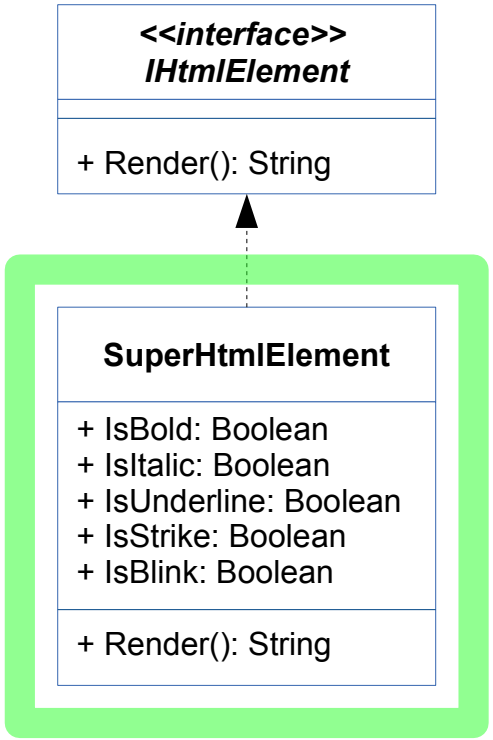
Definition

Refaktorisierungs-Beispiel

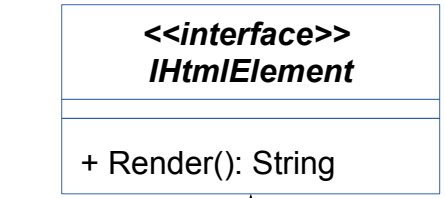
Ausblick







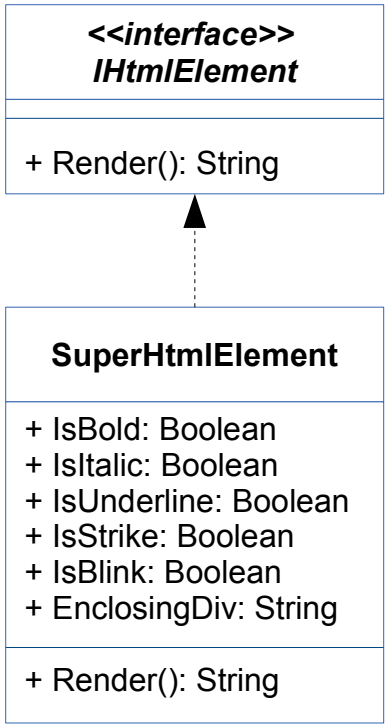

```
function SuperHtmlElement.Render: String
begin
  Result := Self.Text;
  if IsBold then Result := '<b>' + Result + '</b>';
  if IsItalic then Result := '<i>' + Result + '</i>';
  if IsUnderline then Result := '<u>' + Result + '</u>';
  if IsStrike then Result := '<s>' + Result + '</s>';
  if IsBlink then Result := '<blink>' + Result + '</blink>';
end;
```

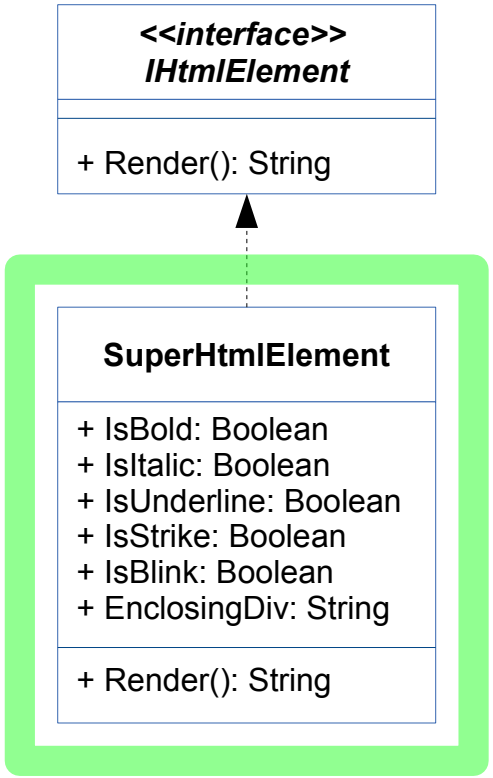


Auch gut.

Können Sie noch ein umschließendes DIV-Element einbauen, bei dem wir den Namen setzen können?

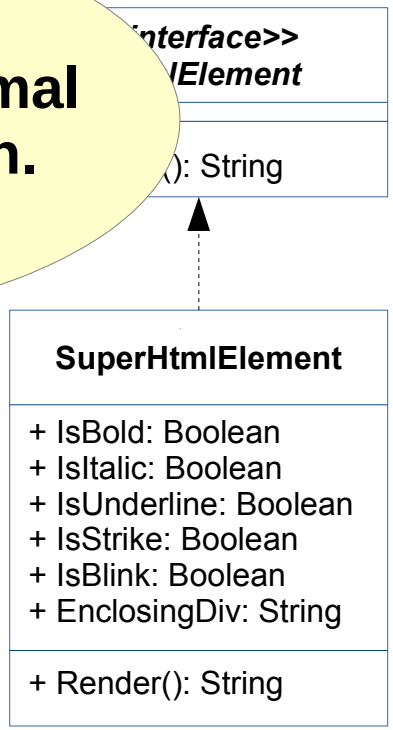
...

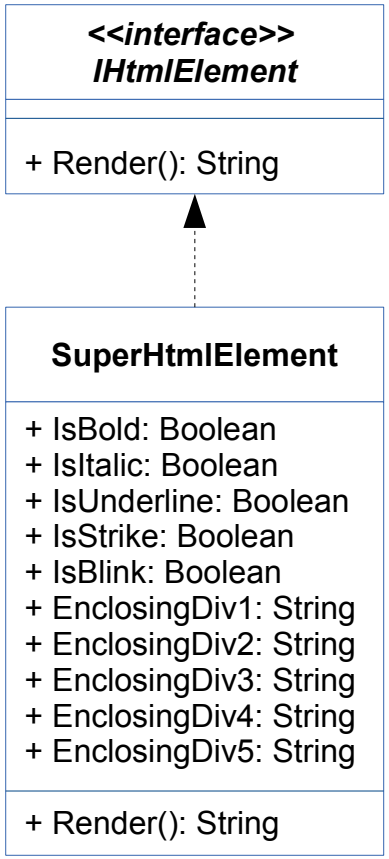


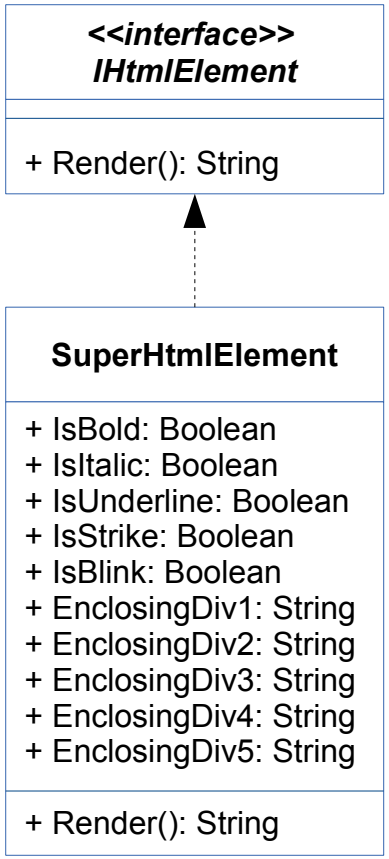


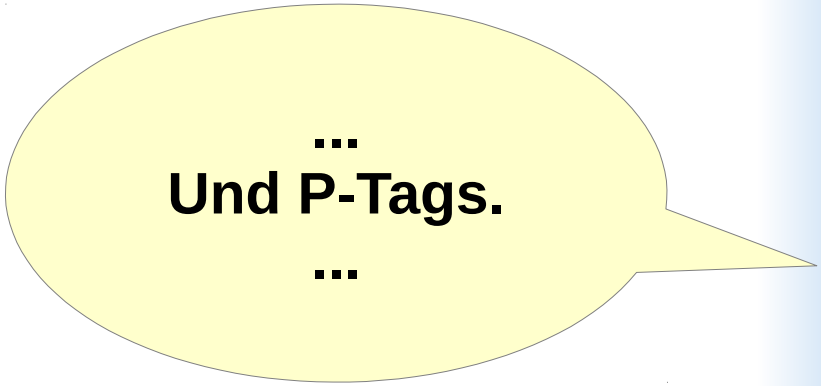

```
function SuperHtmlElement.Render: String
begin
  Result := Self.Text;

  (* ... *)

  if EnclosingDiv <> '' then
    Result := '<div id="' + EnclosingDiv + '">' + Result + '</div>';
end;
```

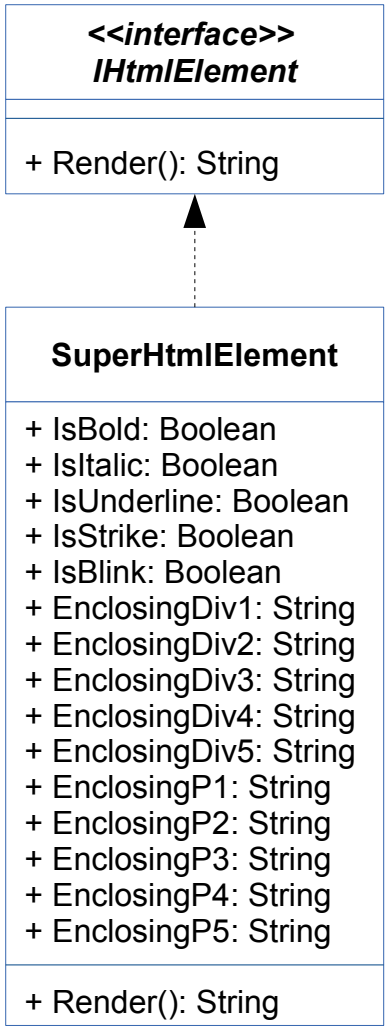
...
Das können aber auch mal drei oder fünf DIV sein.
...



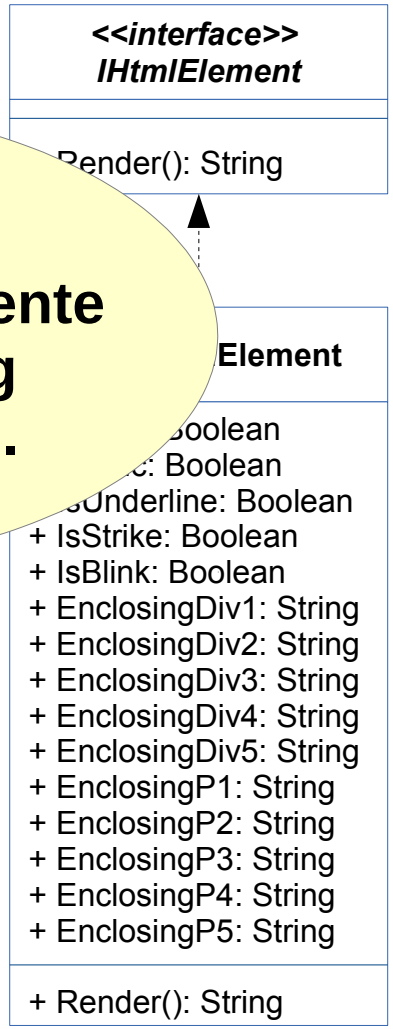

<<interface>>
IHtmlElement
+ Render(): String
SuperHtmlElement
+ IsBold: Boolean
+ IsItalic: Boolean
+ IsUnderline: Boolean
+ IsStrike: Boolean
+ IsBlink: Boolean
+ EnclosingDiv1: String
+ EnclosingDiv2: String
+ EnclosingDiv3: String
+ EnclosingDiv4: String
+ EnclosingDiv5: String
+ Render(): String

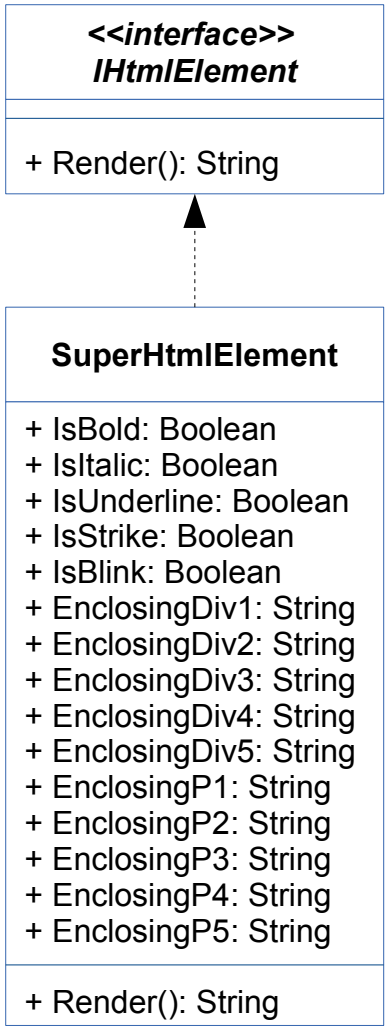

<<interface>>
IHtmlElement
+ Render(): String
SuperHtmlElement
+ IsBold: Boolean
+ IsItalic: Boolean
+ IsUnderline: Boolean
+ IsStrike: Boolean
+ IsBlink: Boolean
+ EnclosingDiv1: String
+ EnclosingDiv2: String
+ EnclosingDiv3: String
+ EnclosingDiv4: String
+ EnclosingDiv5: String
+ Render(): String


...
Und P-Tags.
...


<<interface>>
IHtmlElement
+ Render(): String
SuperHtmlElement
+ IsBold: Boolean
+ IsItalic: Boolean
+ IsUnderline: Boolean
+ IsStrike: Boolean
+ IsBlink: Boolean
+ EnclosingDiv1: String
+ EnclosingDiv2: String
+ EnclosingDiv3: String
+ EnclosingDiv4: String
+ EnclosingDiv5: String
+ EnclosingP1: String
+ EnclosingP2: String
+ EnclosingP3: String
+ EnclosingP4: String
+ EnclosingP5: String
+ Render(): String

Die Reihenfolge der Elemente muss dabei beliebig konfigurierbar sein.




<<interface>>
IHtmlElement
+ Render(): String
SuperHtmlElement
+ IsBold: Boolean
+ IsItalic: Boolean
+ IsUnderline: Boolean
+ IsStrike: Boolean
+ IsBlink: Boolean
+ EnclosingDiv1: String
+ EnclosingDiv2: String
+ EnclosingDiv3: String
+ EnclosingDiv4: String
+ EnclosingDiv5: String
+ EnclosingP1: String
+ EnclosingP2: String
+ EnclosingP3: String
+ EnclosingP4: String
+ EnclosingP5: String
+ Render(): String

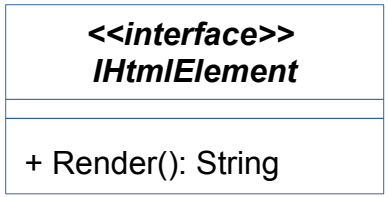
- Fallstudie
  - Lösungsansatz: Klassische Vererbung
  - Lösungsansatz: Objektattribute
  - **Lösungsansatz: Decorator**
- Definition
- Refaktorisierungs-Beispiel
- Ausblick

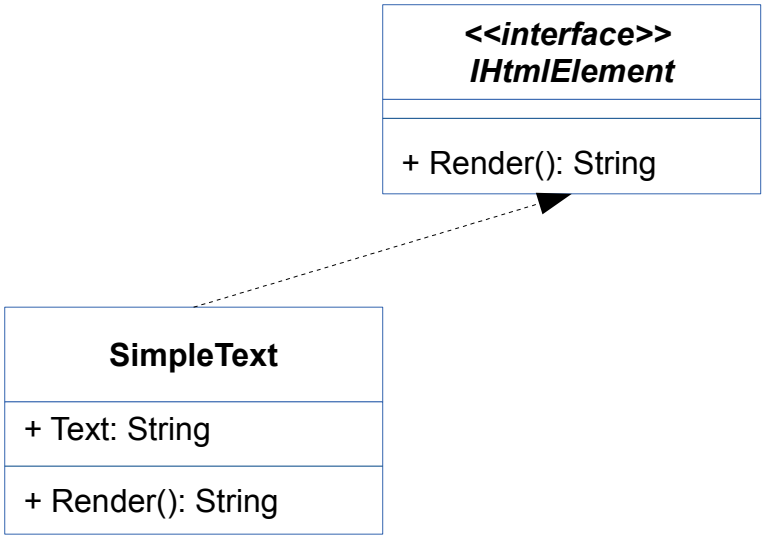
Hallo Welt!

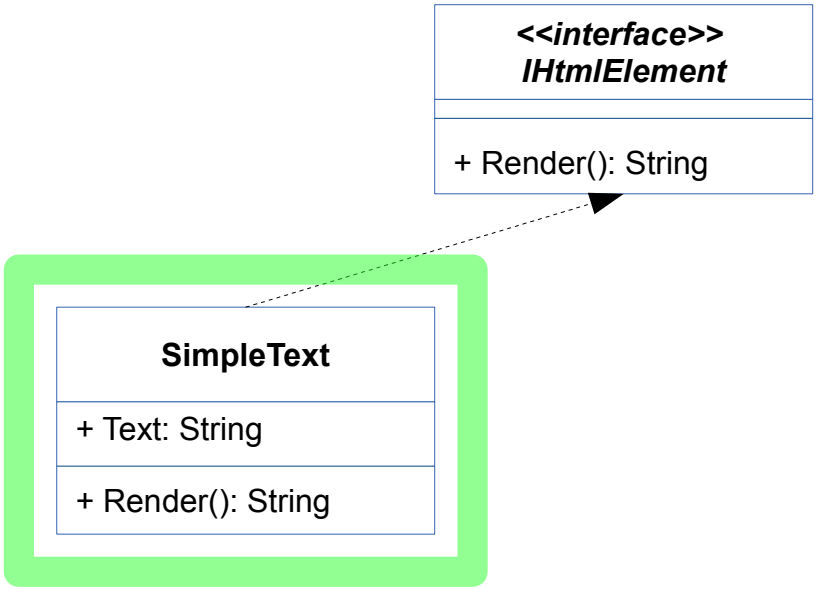
Hallo Welt!

Hallo Welt!

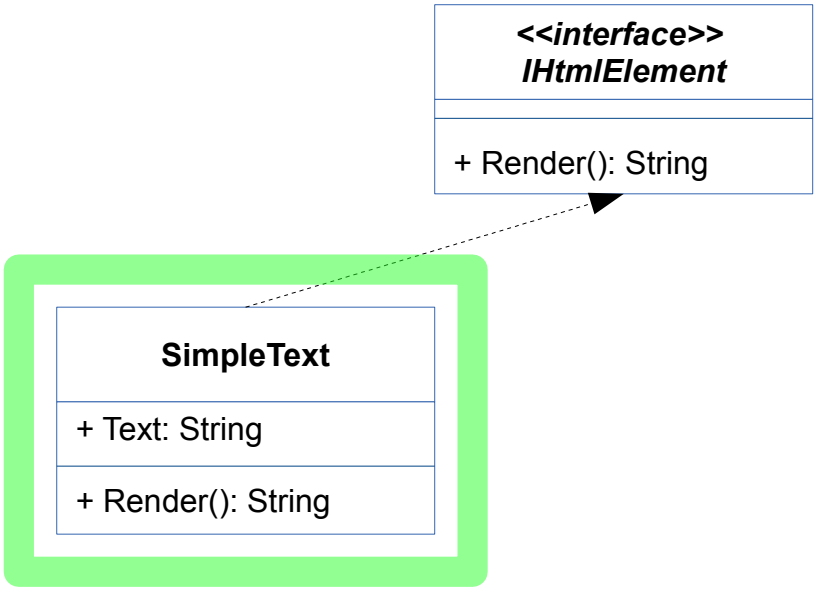
<u>Hallo Welt!</u>



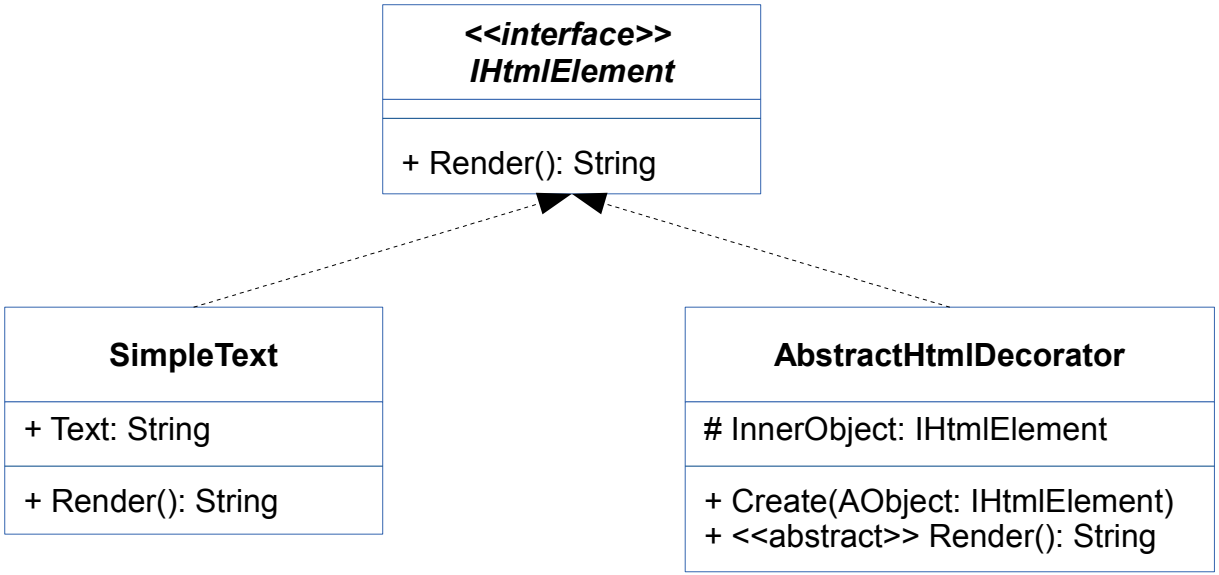

<<interface>>
IHtmlElement
+ Render(): String
SimpleText
+ Text: String
+ Render(): String

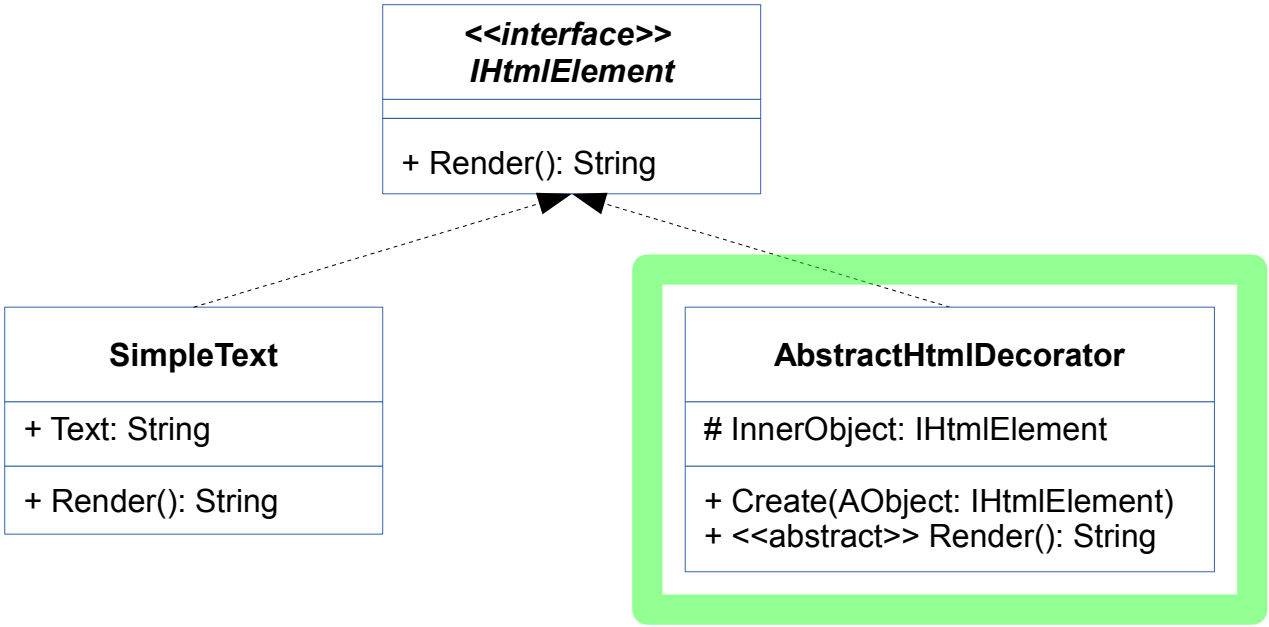



```
TSimpleText = class(IHtmlElement)
  public
    Text: String;
  public
    function Render: String;
end;
```

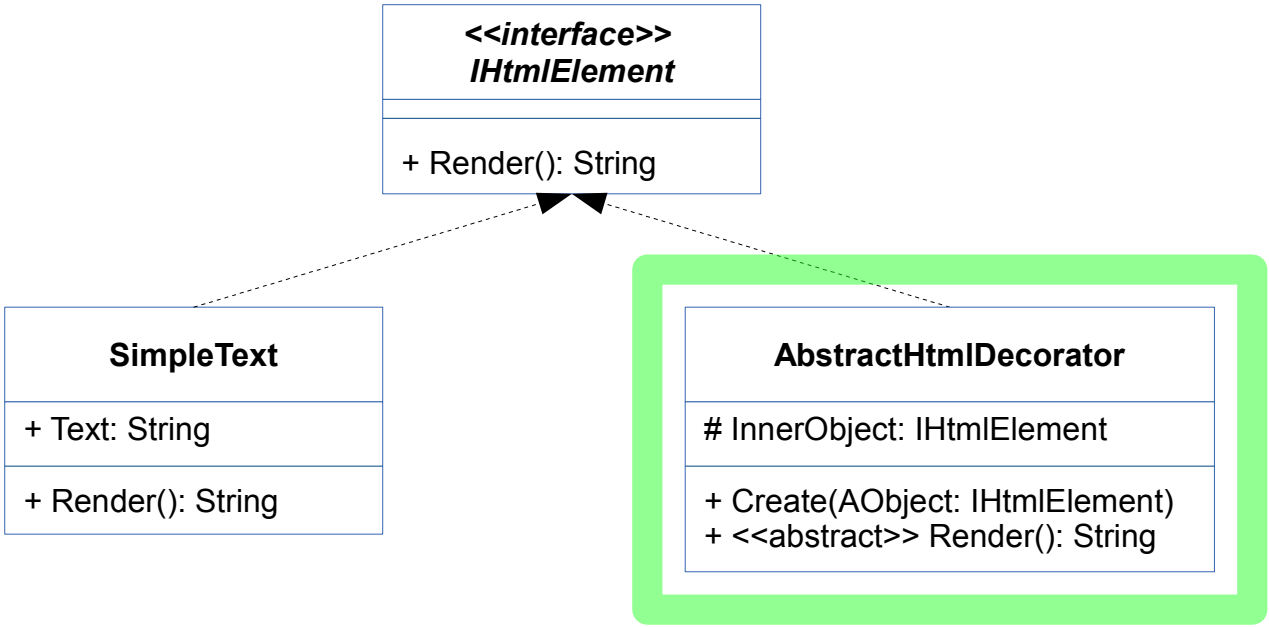



```
function TSimpleText.Render: String;
begin
  Result := Self.Text;
end;
```

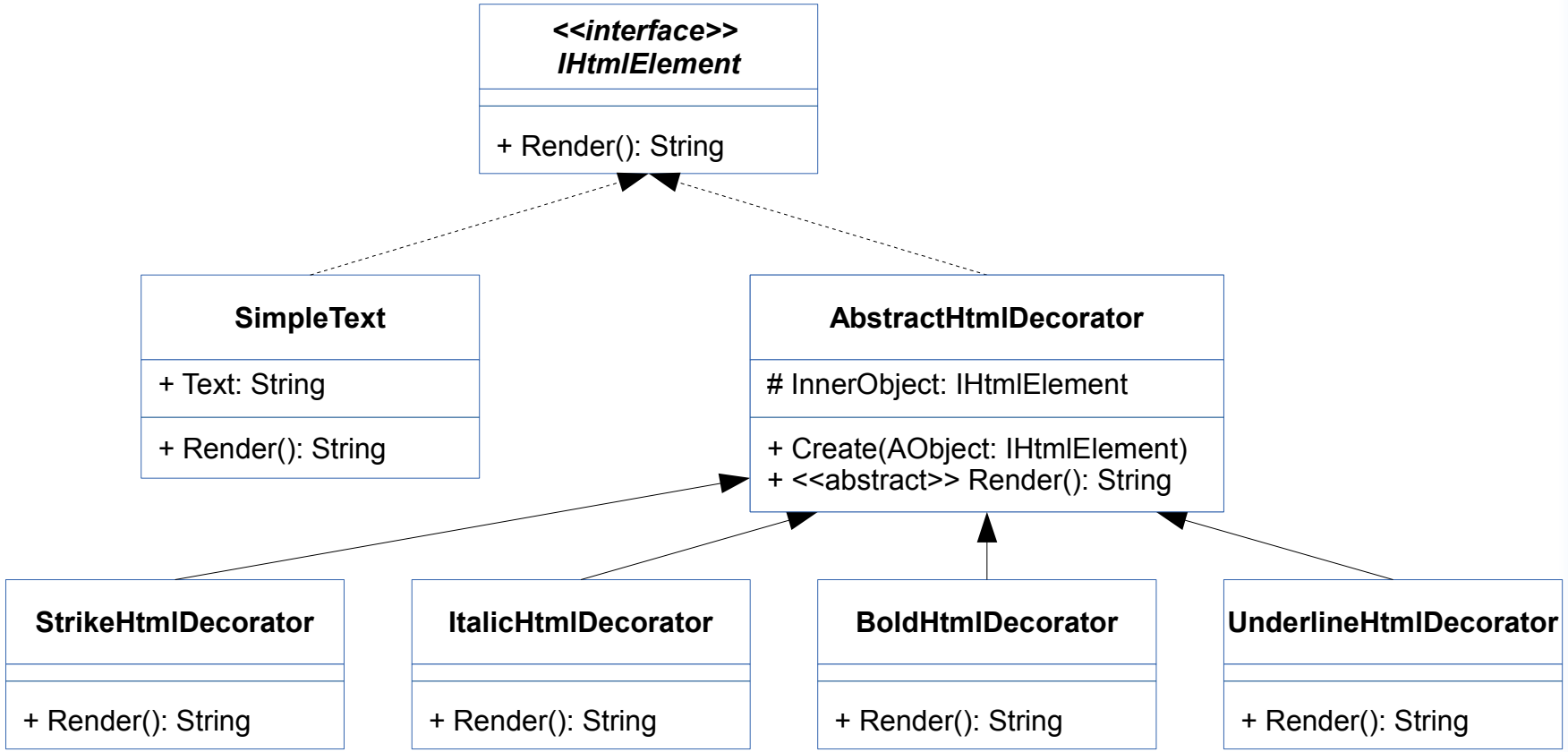

<<interface>>
IHtmlElement
+ Render(): String
SimpleText
+ Text: String
+ Render(): String
AbstractHtmlDecorator
InnerObject: IHtmlElement
+ Create(AObject: IHtmlElement)
+ <<abstract>> Render(): String

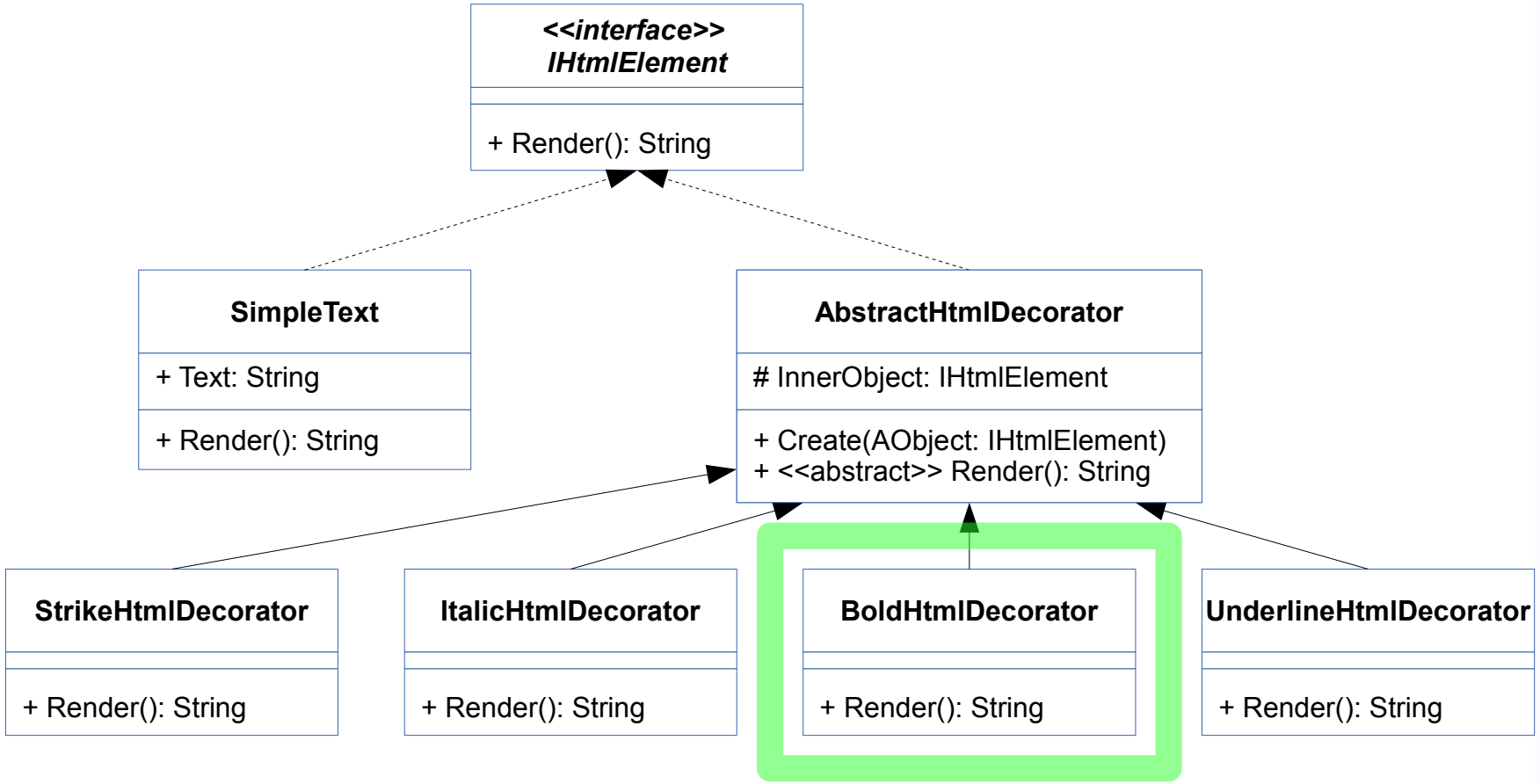



```
TAbstractHtmlDecorator = class(IHtmlElement)
  protected
    InnerObject: IHtmlElement;
  public
    constructor Create(AObject: IHtmlElement);
  public
    function Render: String; virtual abstract;
end;
```

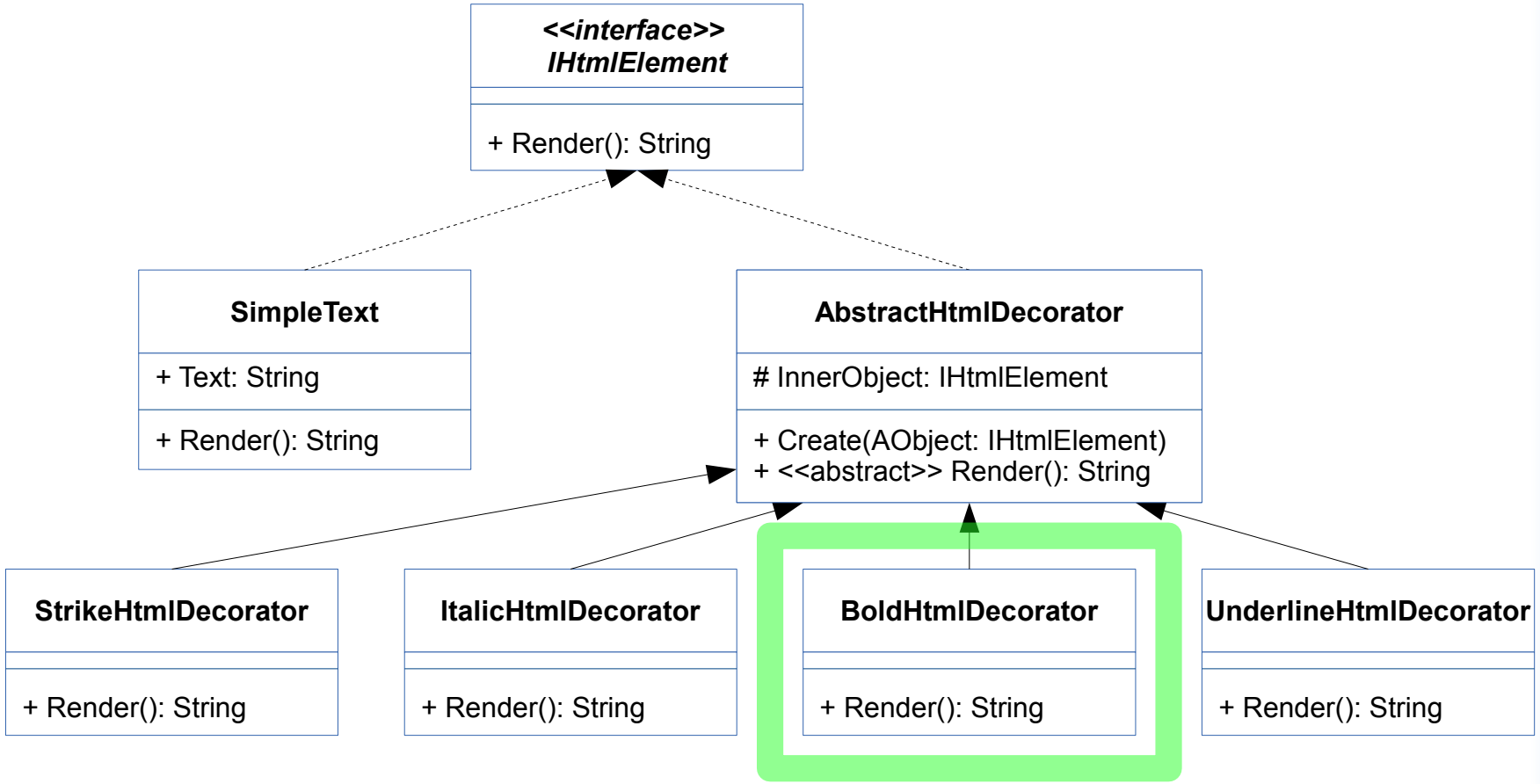



```
constructor TAbstractHtmlDecorator.Create(AObject: IHtmlElement);
begin
  inherited Create;
  InnerObject := AObject;
end;
```






```
TBoldDecorator = class(TAbstractHtmlDecorator)
  public
    function Render: String; override;
end;
```




```
function TBoldDecorator.Render: String;
begin
  Result := '<b>' + InnerObject.Render + '</b>';
end;
```

Beispielaufruf

Beispielaufruf

```
var
  TextObj: TSimpleText;
  DecoratedObj: IHtmlElement;
```

Beispielaufruf

```
var
  TextObj: TSimpleText;
  DecoratedObj: IHtmlElement;

(* ... *)

  TextObj := TSimpleText.Create;
```

Beispielaufruf

```
var
  TextObj: TSimpleText;
  DecoratedObj: IHtmlElement;

(* ... *)

  TextObj := TSimpleText.Create;
  TextObj.Text := 'Hallo Welt!';
```

Beispielaufruf

```
var
  TextObj: TSimpleText;
  DecoratedObj: IHtmlElement;

(* ... *)

  TextObj := TSimpleText.Create;
  TextObj.Text := 'Hallo Welt!';

  DecoratedObj := TUnderlineDecorator.Create(TextObj);
```

Beispielaufruf

```
var
  TextObj: TSimpleText;
  DecoratedObj: IHtmlElement;

(* ... *)

  TextObj := TSimpleText.Create;
  TextObj.Text := 'Hallo Welt!';

  DecoratedObj := TUnderlineDecorator.Create(TextObj);
  DecoratedObj := TBoldDecorator.Create(DecoratedObj);
```

Beispielaufruf

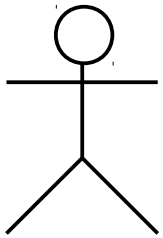
```
var
  TextObj: TSimpleText;
  DecoratedObj: IHtmlElement;

(* ... *)

  TextObj := TSimpleText.Create;
  TextObj.Text := 'Hallo Welt!';

  DecoratedObj := TUnderlineDecorator.Create(TextObj);
  DecoratedObj := TBoldDecorator.Create(DecoratedObj);
  DecoratedObj := TItalicDecorator.Create(DecoratedObj);
```

Beispielaufruf

```
var
  TextObj: TSimpleText;
  DecoratedObj: IHtmlElement;

(* ... *)

  TextObj := TSimpleText.Create;
  TextObj.Text := 'Hallo Welt!';

  DecoratedObj := TUnderlineDecorator.Create(TextObj);
  DecoratedObj := TBoldDecorator.Create(DecoratedObj);
  DecoratedObj := TItalicDecorator.Create(DecoratedObj);

  WriteLn(DecoratedObj.Render);
```

Beispielaufruf

```
var
  TextObj: TSimpleText;
  DecoratedObj: IHtmlElement;

(* ... *)

  TextObj := TSimpleText.Create;
  TextObj.Text := 'Hallo Welt!';

  DecoratedObj := TUnderlineDecorator.Create(TextObj);
  DecoratedObj := TBoldDecorator.Create(DecoratedObj);
  DecoratedObj := TItalicDecorator.Create(DecoratedObj);

  WriteLn(DecoratedObj.Render);
```



ItalicDecorator

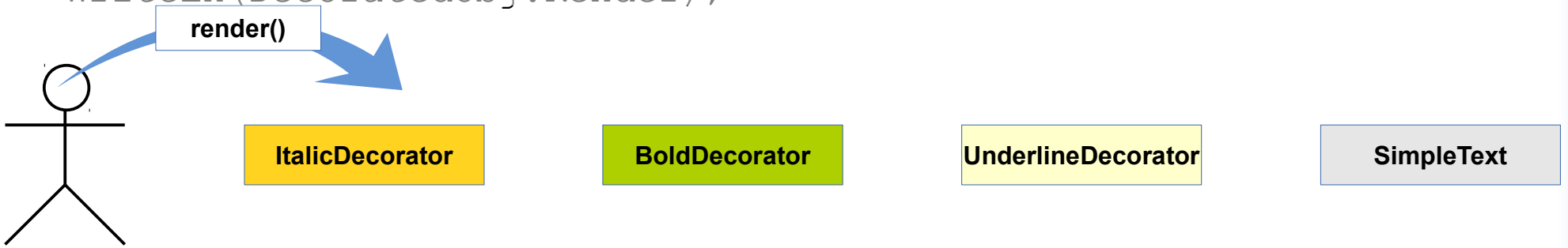
BoldDecorator

UnderlineDecorator

SimpleText

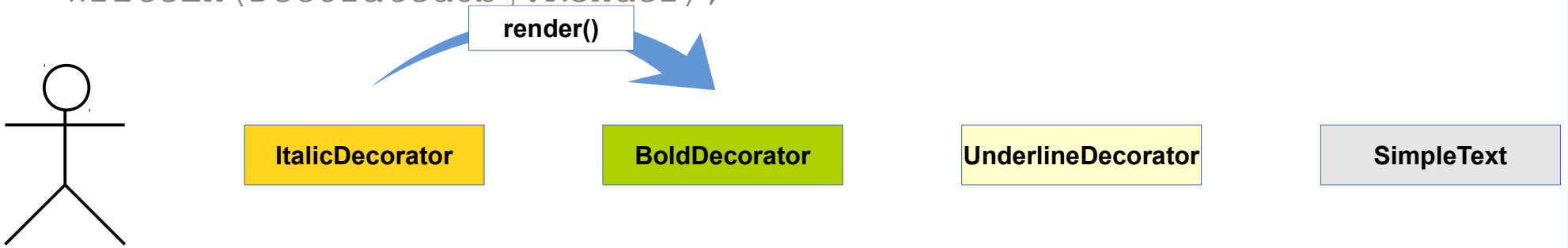
Beispielaufruf

```
var
  TextObj: TSimpleText;
  DecoratedObj: IHtmlElement;

(* ... *)

  TextObj := TSimpleText.Create;
  TextObj.Text := 'Hallo Welt!';

  DecoratedObj := TUnderlineDecorator.Create(TextObj);
  DecoratedObj := TBoldDecorator.Create(DecoratedObj);
  DecoratedObj := TItalicDecorator.Create(DecoratedObj);

  WriteLn(DecoratedObj.Render);
```



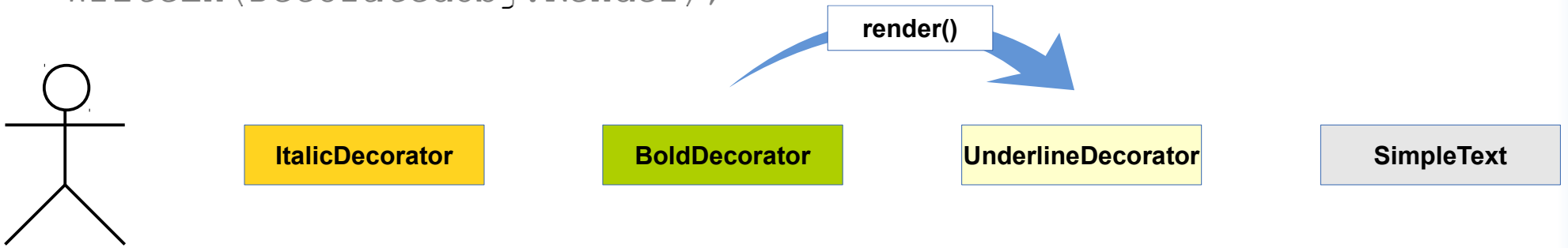
Beispielaufruf

```
var
  TextObj: TSimpleText;
  DecoratedObj: IHtmlElement;

(* ... *)

  TextObj := TSimpleText.Create;
  TextObj.Text := 'Hallo Welt!';

  DecoratedObj := TUnderlineDecorator.Create(TextObj);
  DecoratedObj := TBoldDecorator.Create(DecoratedObj);
  DecoratedObj := TItalicDecorator.Create(DecoratedObj);

  WriteLn(DecoratedObj.Render);
```



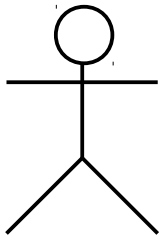
Beispielaufruf

```
var
  TextObj: TSimpleText;
  DecoratedObj: IHtmlElement;

(* ... *)

  TextObj := TSimpleText.Create;
  TextObj.Text := 'Hallo Welt!';

  DecoratedObj := TUnderlineDecorator.Create(TextObj);
  DecoratedObj := TBoldDecorator.Create(DecoratedObj);
  DecoratedObj := TItalicDecorator.Create(DecoratedObj);

  WriteLn(DecoratedObj.Render);
```



Beispielaufruf

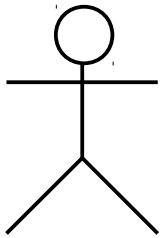
```
var
  TextObj: TSimpleText;
  DecoratedObj: IHtmlElement;

(* ... *)

  TextObj := TSimpleText.Create;
  TextObj.Text := 'Hallo Welt!';

  DecoratedObj := TUnderlineDecorator.Create(TextObj);
  DecoratedObj := TBoldDecorator.Create(DecoratedObj);
  DecoratedObj := TItalicDecorator.Create(DecoratedObj);

  WriteLn(DecoratedObj.Render);
```

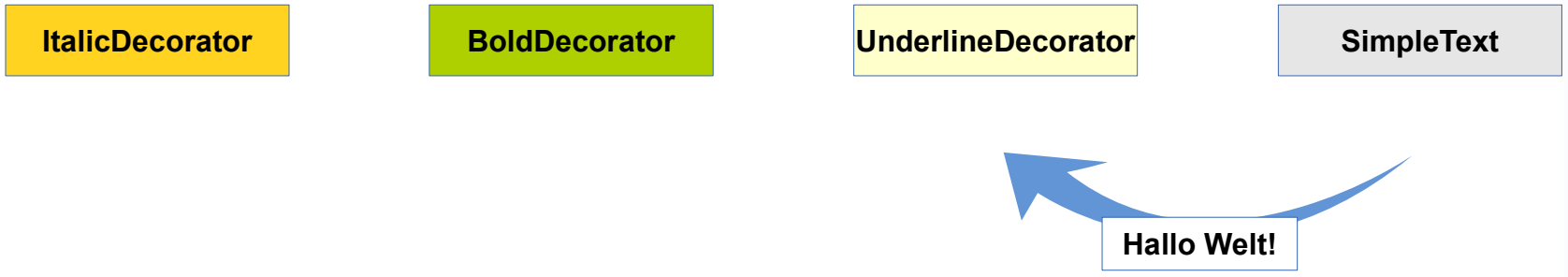


render()

Beispielaufruf

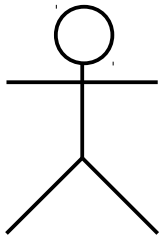
```
var
  TextObj: TSimpleText;
  DecoratedObj: IHtmlElement;

(* ... *)

  TextObj := TSimpleText.Create;
  TextObj.Text := 'Hallo Welt!';

  DecoratedObj := TUnderlineDecorator.Create(TextObj);
  DecoratedObj := TBoldDecorator.Create(DecoratedObj);
  DecoratedObj := TItalicDecorator.Create(DecoratedObj);

  WriteLn(DecoratedObj.Render);
```

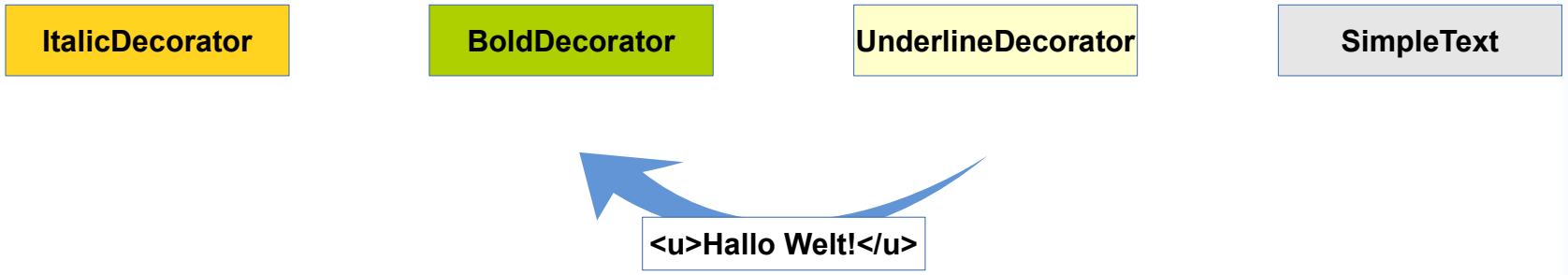




Beispielaufruf

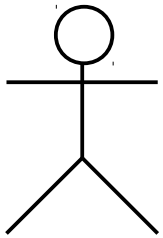
```
var
  TextObj: TSimpleText;
  DecoratedObj: IHtmlElement;

(* ... *)

  TextObj := TSimpleText.Create;
  TextObj.Text := 'Hallo Welt!';

  DecoratedObj := TUnderlineDecorator.Create(TextObj);
  DecoratedObj := TBoldDecorator.Create(DecoratedObj);
  DecoratedObj := TItalicDecorator.Create(DecoratedObj);

  WriteLn(DecoratedObj.Render);
```

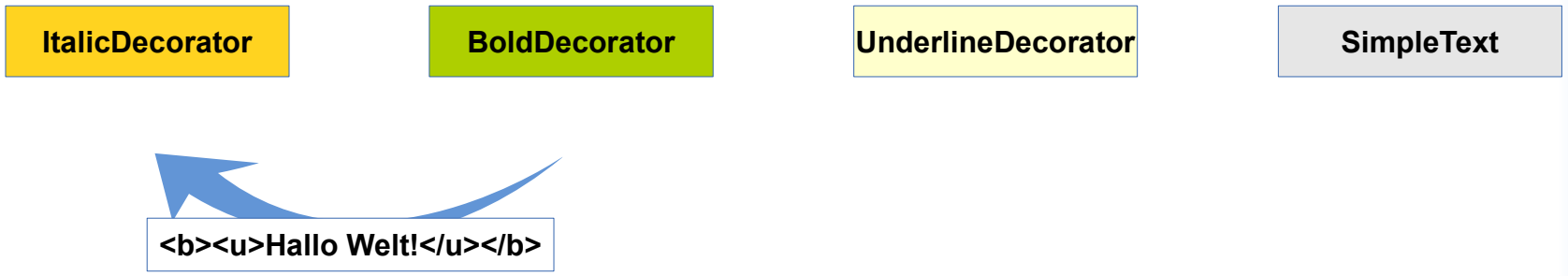




Beispielaufruf

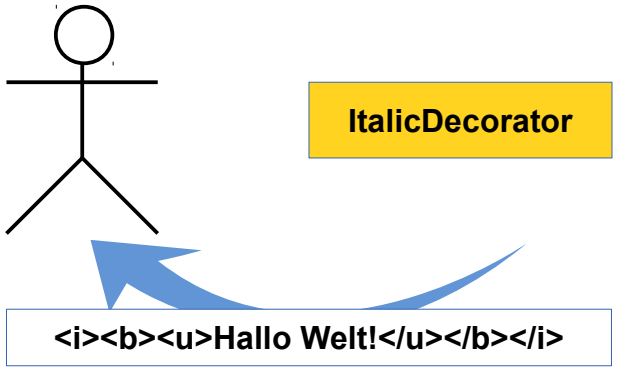
```
var
  TextObj: TSimpleText;
  DecoratedObj: IHtmlElement;

(* ... *)

  TextObj := TSimpleText.Create;
  TextObj.Text := 'Hallo Welt!';

  DecoratedObj := TUnderlineDecorator.Create(TextObj);
  DecoratedObj := TBoldDecorator.Create(DecoratedObj);
  DecoratedObj := TItalicDecorator.Create(DecoratedObj);

  WriteLn(DecoratedObj.Render);
```





Beispielaufruf

```
var
  TextObj: TSimpleText;
  DecoratedObj: IHtmlElement;

(* ... *)

  TextObj := TSimpleText.Create;
  TextObj.Text := 'Hallo Welt!';

  DecoratedObj := TUnderlineDecorator.Create(TextObj);
  DecoratedObj := TBoldDecorator.Create(DecoratedObj);
  DecoratedObj := TItalicDecorator.Create(DecoratedObj);

  WriteLn(DecoratedObj.Render);
```

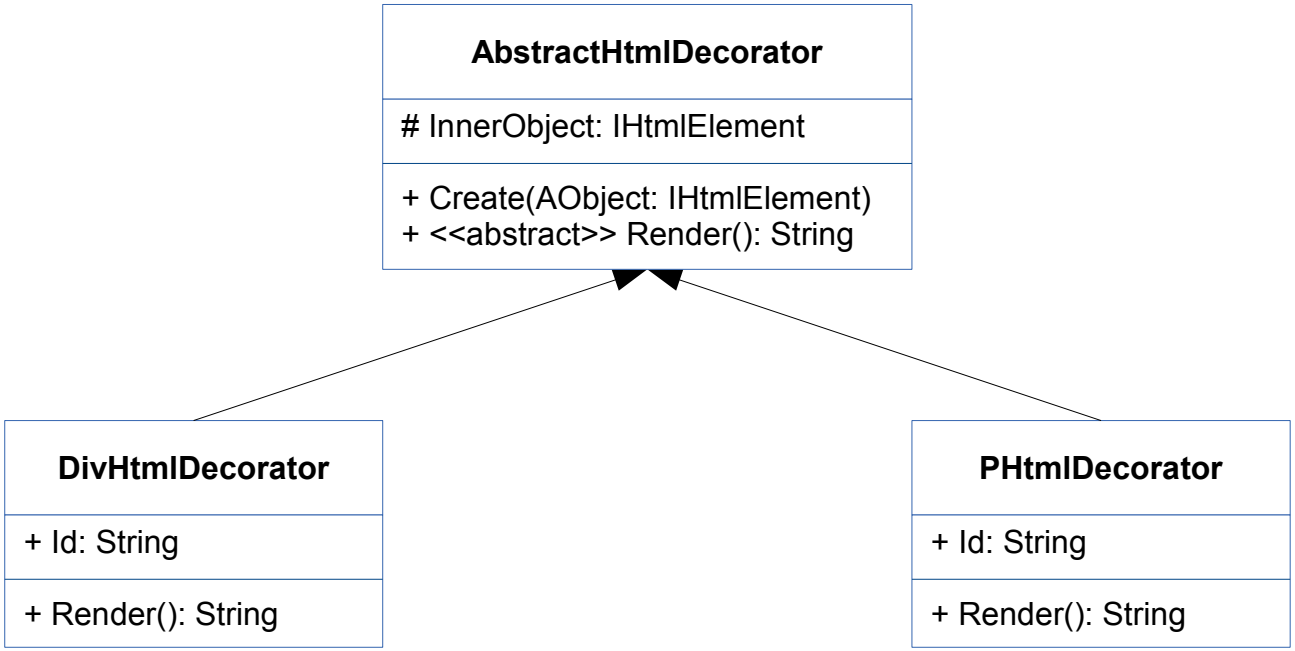


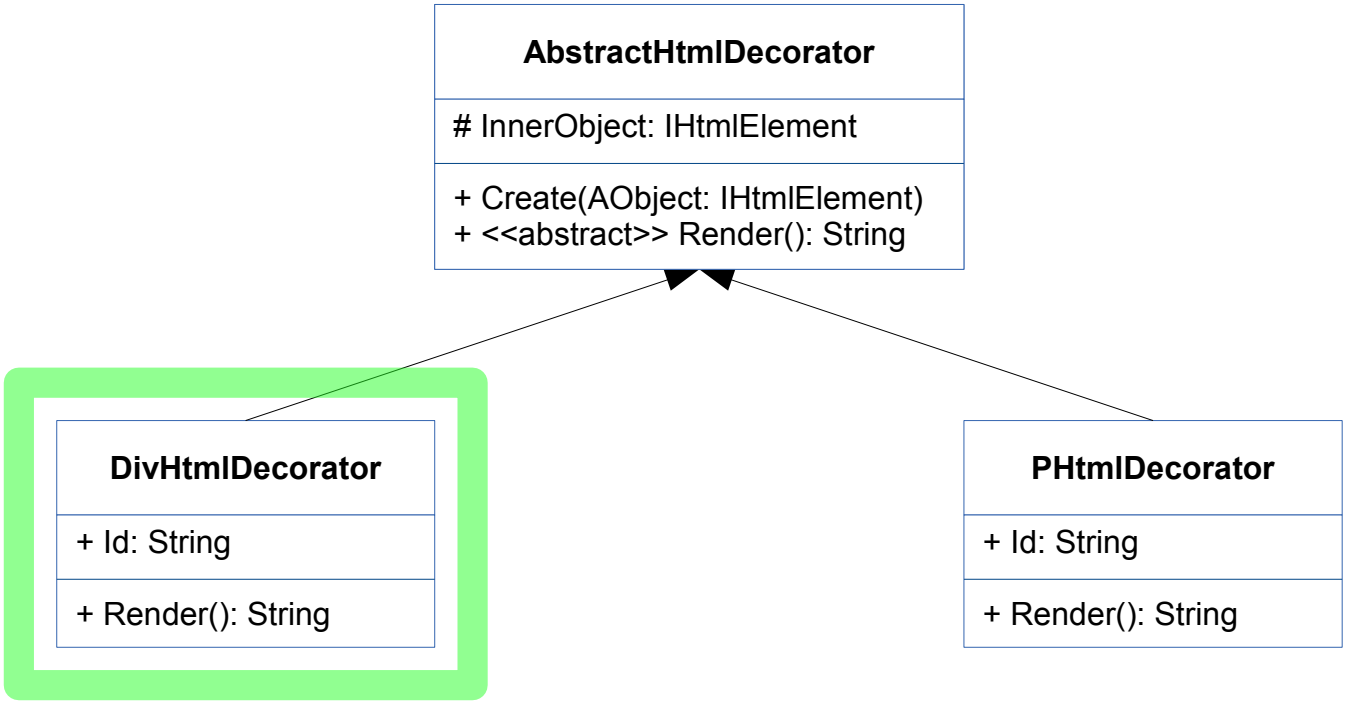
Beispielaufruf

```
var
  TextObj: TSimpleText;
  DecoratedObj: IHtmlElement;

(* ... *)

  TextObj := TSimpleText.Create;
  TextObj.Text := 'Hallo Welt!';

  DecoratedObj := TUnderlineDecorator.Create(TextObj);
  DecoratedObj := TBoldDecorator.Create(DecoratedObj);
  DecoratedObj := TItalicDecorator.Create(DecoratedObj);

  WriteLn(DecoratedObj.Render);
```

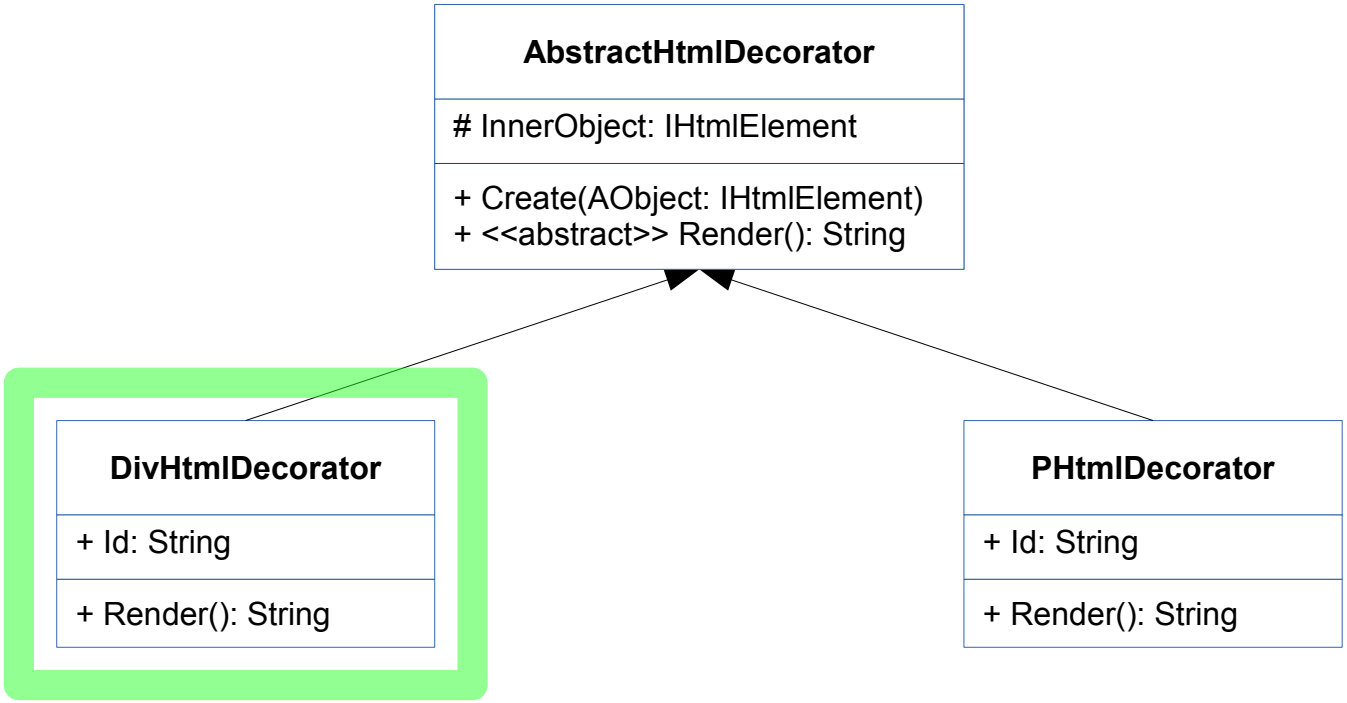
Ausgabe:

```
<i><b><u>Hallo Welt!</u></b></i>
```


AbstractHtmlDecorator
InnerObject: IHtmlElement
+ Create(AObject: IHtmlElement)
+ <<abstract>> Render(): String
DivHtmlDecorator
+ Id: String
+ Render(): String
PHtmlDecorator
+ Id: String
+ Render(): String




```
TDivHtmlDecorator = class(TAbstractHtmlDecorator)
  private
    Id: String;
  public
    constructor Create(AObject: IHtmlElement; AId: String);
    function Render: String; override;
end;
```




```
constructor TDivHtmlDecorator.Create(AObject: IHtmlElement;
                                     AId: String);
begin
  inherited Create(AObject);
  Id := AId;
end;

function TDivHtmlDecorator.Render: String;
begin
  Result := '<div id="' + Id + '">' + InnerObject.Render + '</div>';
end;
```

Beispielaufruf

```
var
  TextObj2: TSimpleText;
  DecoratedObj2: IHtmlElement;

(* ... *)

TextObj2 := TSimpleText.Create;
TextObj2.Text := 'Hallo Welt!';

DecoratedObj2 := TDivHtmlDecorator.Create(TextObj2, 'My1');
DecoratedObj2 := TPHtmlDecorator.Create(DecoratedObj2, 'My2');
DecoratedObj2 := TDivHtmlDecorator.Create(DecoratedObj2, 'My3');

WriteLn(DecoratedObj2.Render);
```

Beispielaufruf

```
var
  TextObj2: TSimpleText;
  DecoratedObj2: IHtmlElement;

(* ... *)

TextObj2 := TSimpleText.Create;
TextObj2.Text := 'Hallo Welt!';

DecoratedObj2 := TDivHtmlDecorator.Create(TextObj2, 'My1');
DecoratedObj2 := TPHtmlDecorator.Create(DecoratedObj2, 'My2');
DecoratedObj2 := TDivHtmlDecorator.Create(DecoratedObj2, 'My3');

WriteLn(DecoratedObj2.Render);
```

Ausgabe:

```
<div id="My3"><p id="My2"><div id="My1">Hallo Welt!</div></p></div>
```

Fallstudie

Lösungsansatz: Klassische Vererbung

Lösungsansatz: Objektattribute

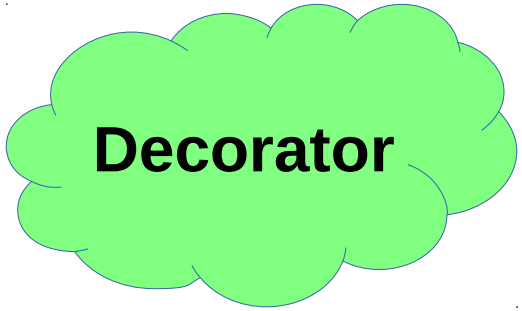
Lösungsansatz: Decorator

Definition

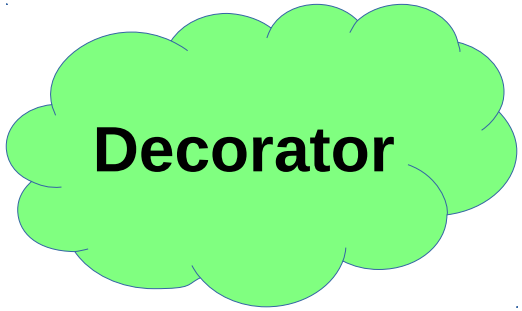
Refaktorisierungs-Beispiel

Ausblick

Definition

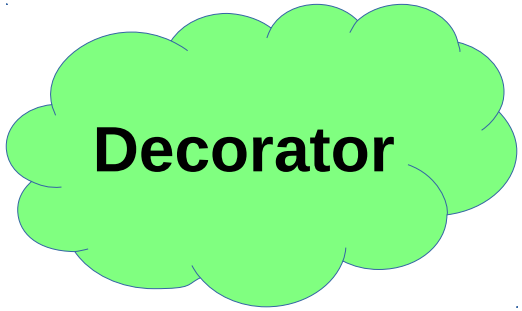


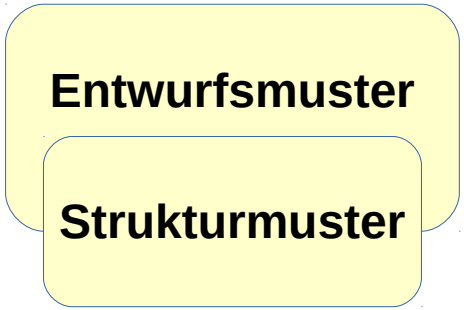
Definition



Entwurfsmuster

Definition





Definition

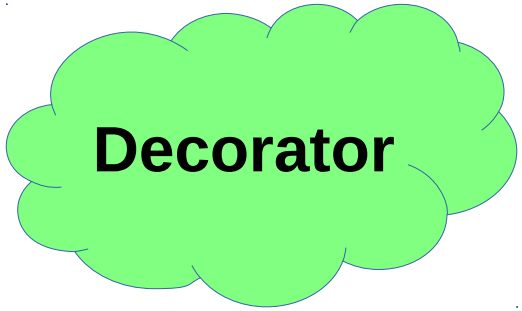
Decorator

Entwurfsmuster

Strukturmuster

dient der Erweiterung einer Klasse mit neuer Funktionalität

Definition



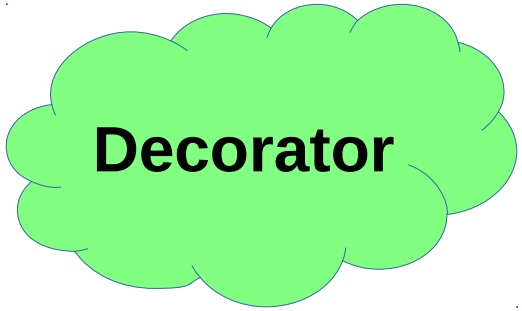
Entwurfsmuster

Strukturmuster

dient der Erweiterung einer Klasse mit neuer Funktionalität

Alternative zur Vererbung

Definition



Entwurfsmuster

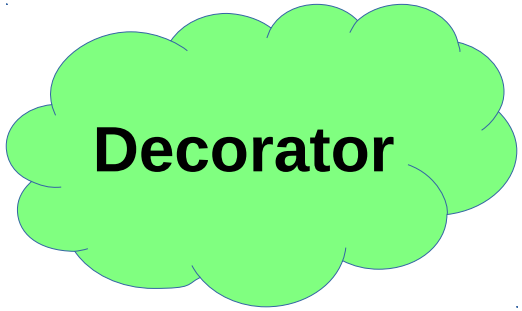
Strukturmuster

dient der Erweiterung einer Klasse mit neuer Funktionalität

Umhüllung des Ursprungsobjektes

Alternative zur Vererbung

Definition



Entwurfsmuster

Strukturmuster

dient der Erweiterung einer Klasse mit neuer Funktionalität

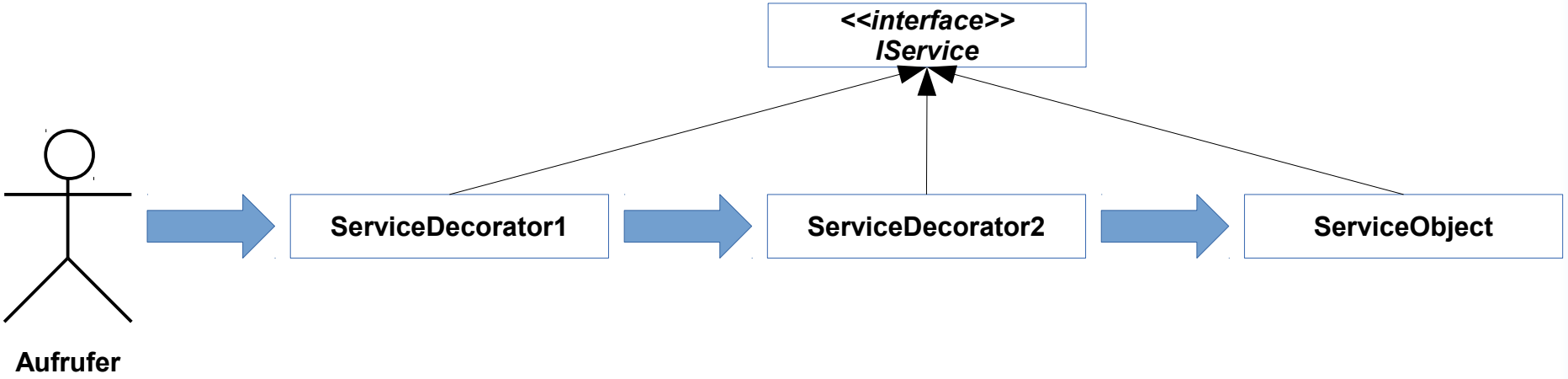
Umhüllung des Ursprungsobjektes

transparente Weiterleitung von Methodenaufrufen

Alternative zur Vererbung

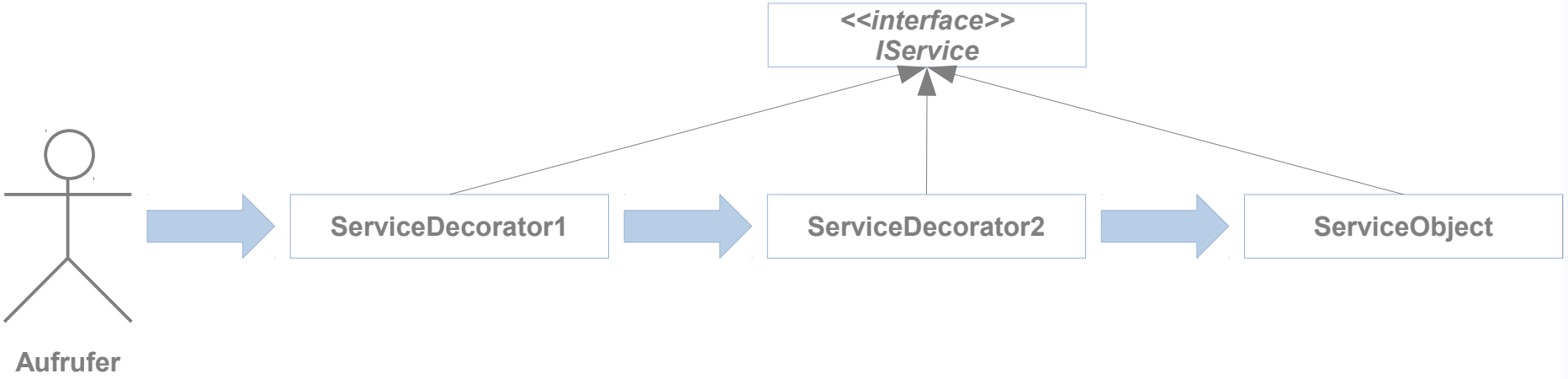
transparente Weiterleitung von Methodenaufrufen

Aufrufpfad

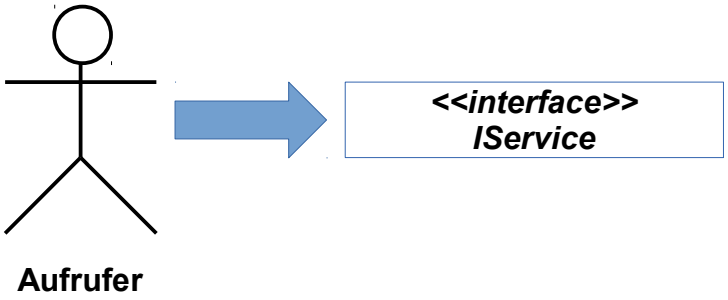


transparente Weiterleitung von Methodenaufrufen

Aufrufpfad



Sicht des Aufrufers



- Fallstudie
  - Lösungsansatz: Klassische Vererbung
  - Lösungsansatz: Objektattribute
  - Lösungsansatz: Decorator
- Definition
- **Refaktorisierungs-Beispiel**
- Ausblick

Refaktorisierungs-Beispiel

Mate24.com

Refaktorisierungs-Beispiel

Mate24.com

MateService
+ OrderMate(Username: String, Password: String, Count: Integer, Product: String): String

Refaktorisierungs-Beispiel

Mate24.com

MateService

+ OrderMate(Username: String, Password: String, Count: Integer, Product: String): String

DbService

+ GetPrice(ProductName: String): Double
+ HasDiscount(Username: String): Boolean
+ GetStock(ProductName: String): Integer
+ SetStock(ProductName: String, Count: Integer)
+ GetUserContingent(Username: String): Double
+ SetUserContingent(Username: String, Contingent: Double)
+ GetUserAdress(Username: String): String

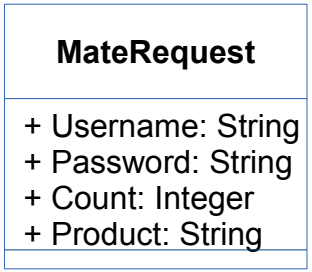
UserService

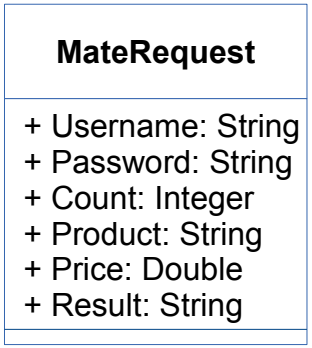
+ Authenticate(Username: String, Password: String): Boolean

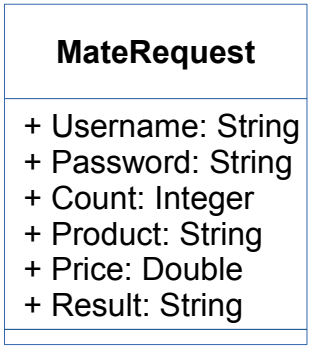
MailService

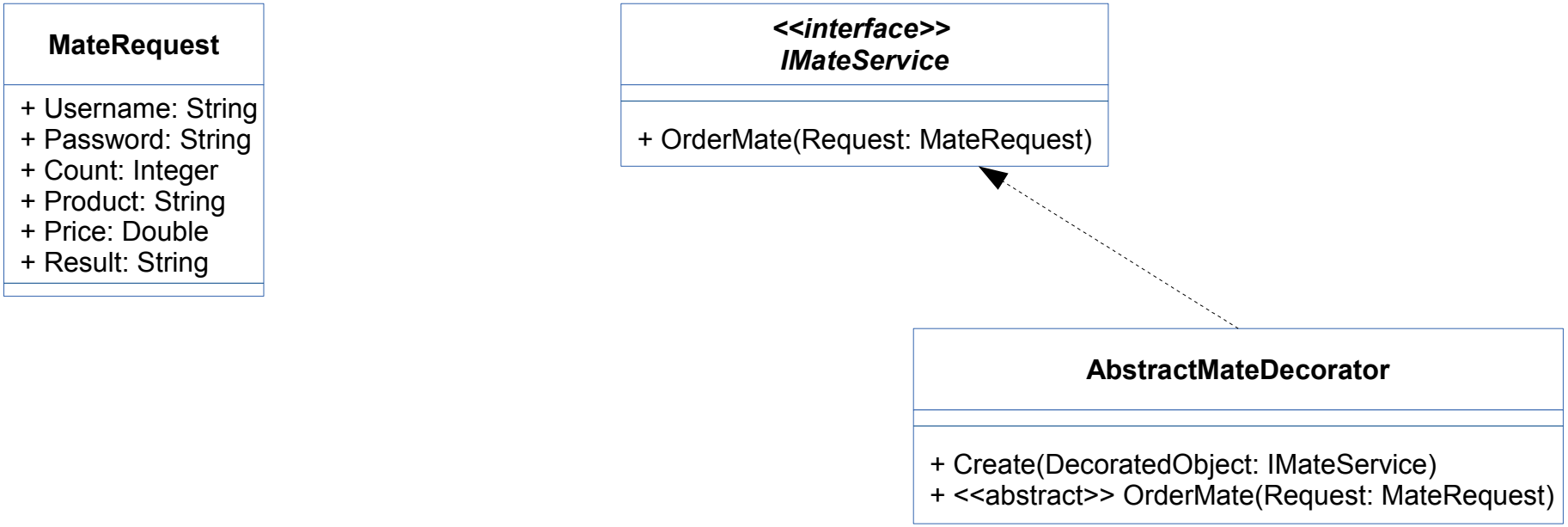
+ SendMail(Receipient: String, Subject: String, MailText: String

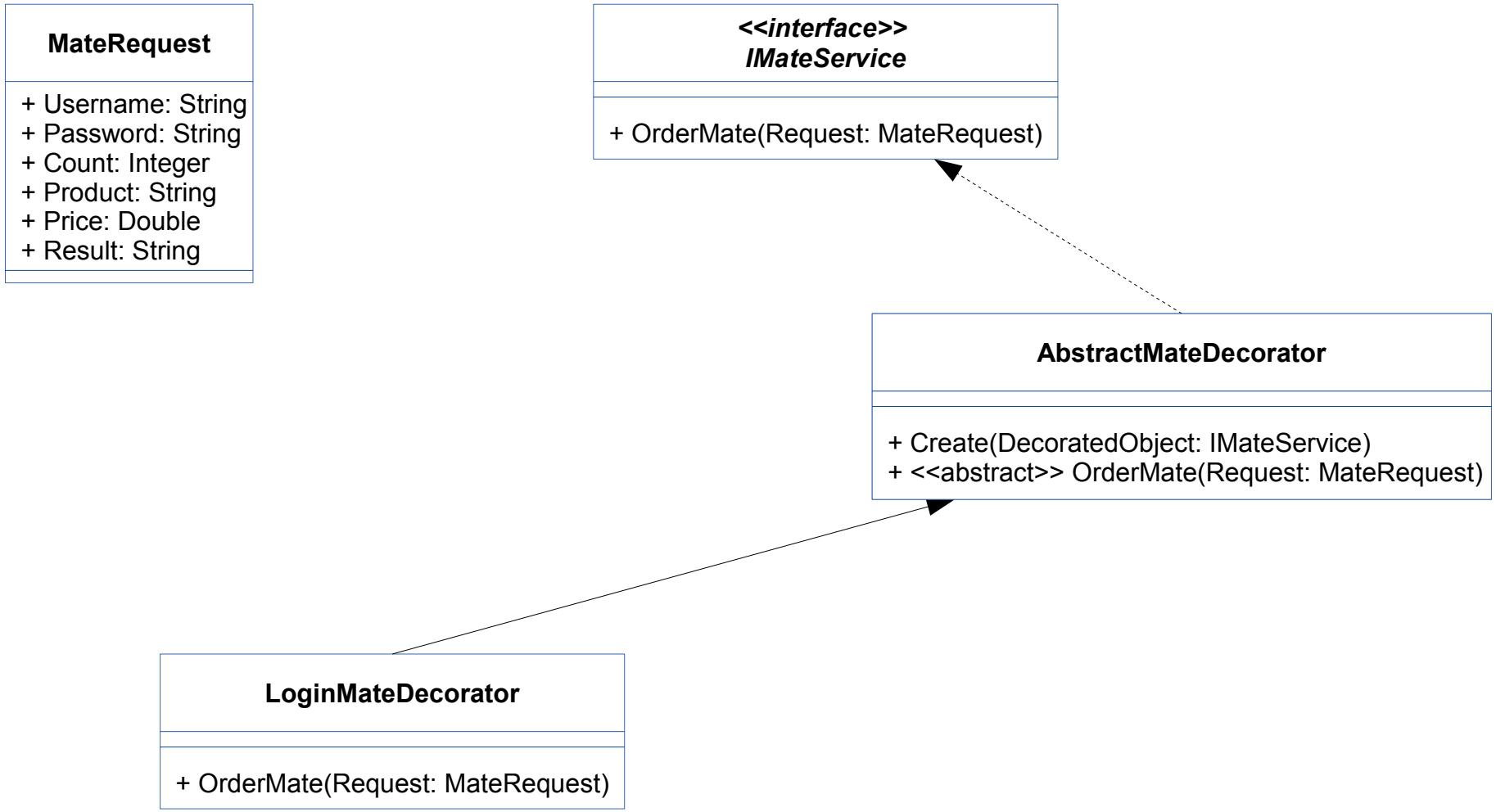
```
function TMateService.OrderMate(Username, Password: String;
                                Count: Integer; Product: String): String;
var
  SinglePrice, Price, Contingent: Double;
  Stock: Integer;
  Address, MailText: String;
begin
  if TUserService.Authenticate(Username, Password) then begin
    SinglePrice := TDbService.GetPrice(Product);
    Price := SinglePrice * Count;
    if TDbService.HasDiscount(Username) then Price := Price * 0.9;
    Contingent := TDbService.GetUserContingent(Username);
    Contingent := Contingent - Price;
    if Contingent >= 0 then begin
      Stock := TDbService.GetStock(Product);
      Stock := Stock - Count;
      if Stock >= 0 then begin
        Address := TDbService.GetUserAdress(Username);
        MailText := 'Liebes Versand-Team, ' + LineEnding
         + 'schicke ' + IntToStr(Count) + 'x ' + Product + ' an:' + LineEnding
         + Username + LineEnding + Address;
        TMailService.SendMail('ship@mate24.de', 'Order', Mailtext);
        TDbService.SetUserContingent(Username, Contingent);
        TDbService.SetStock(Product, Stock);
        Result := 'OK';
      end else
        Result := 'not enough in stock';
    end else
      Result := 'not enough credits';
  end else
    Result := 'login failed';
end;
```

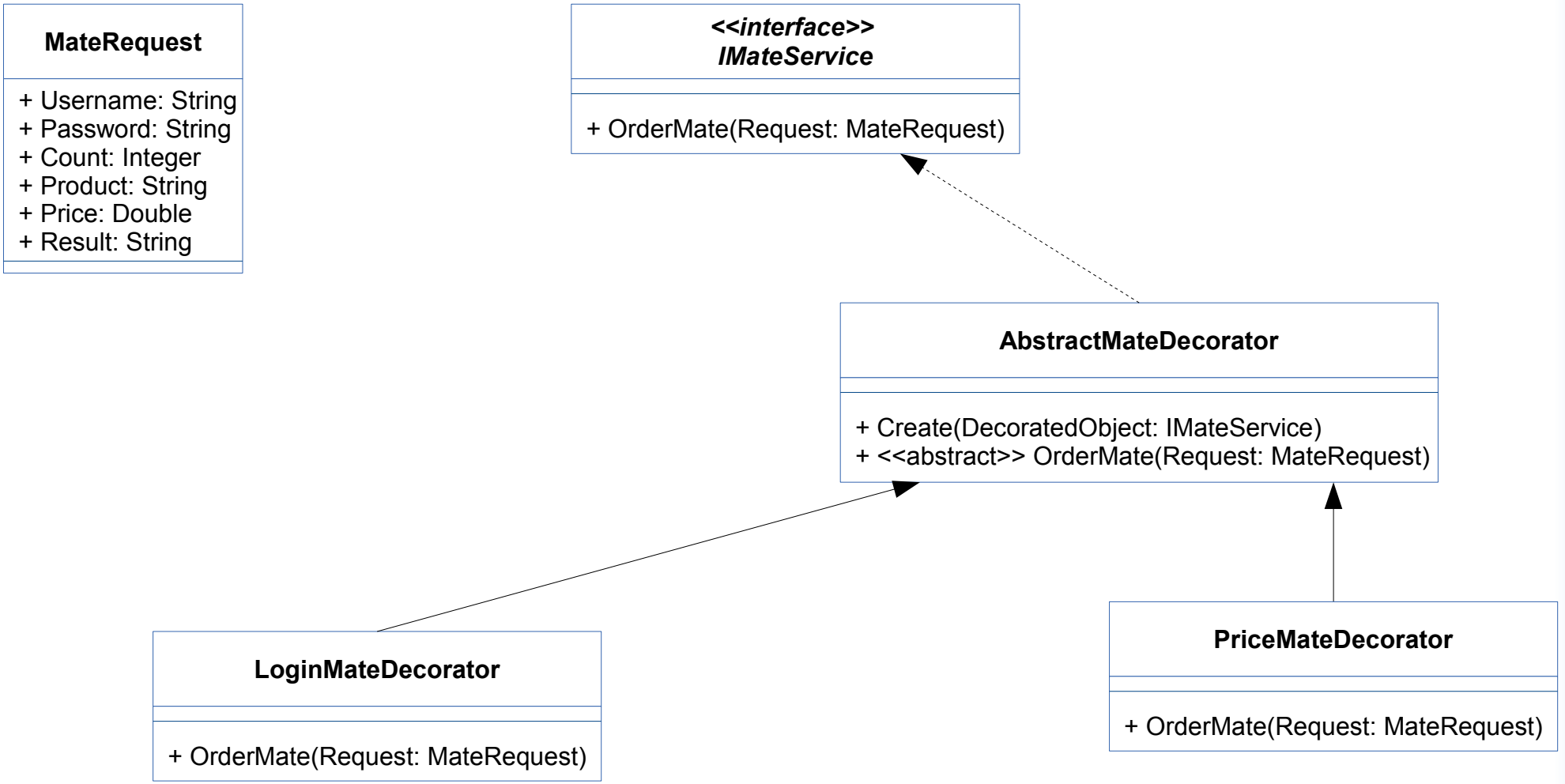

MateRequest
+ Username: String
+ Password: String
+ Count: Integer
+ Product: String

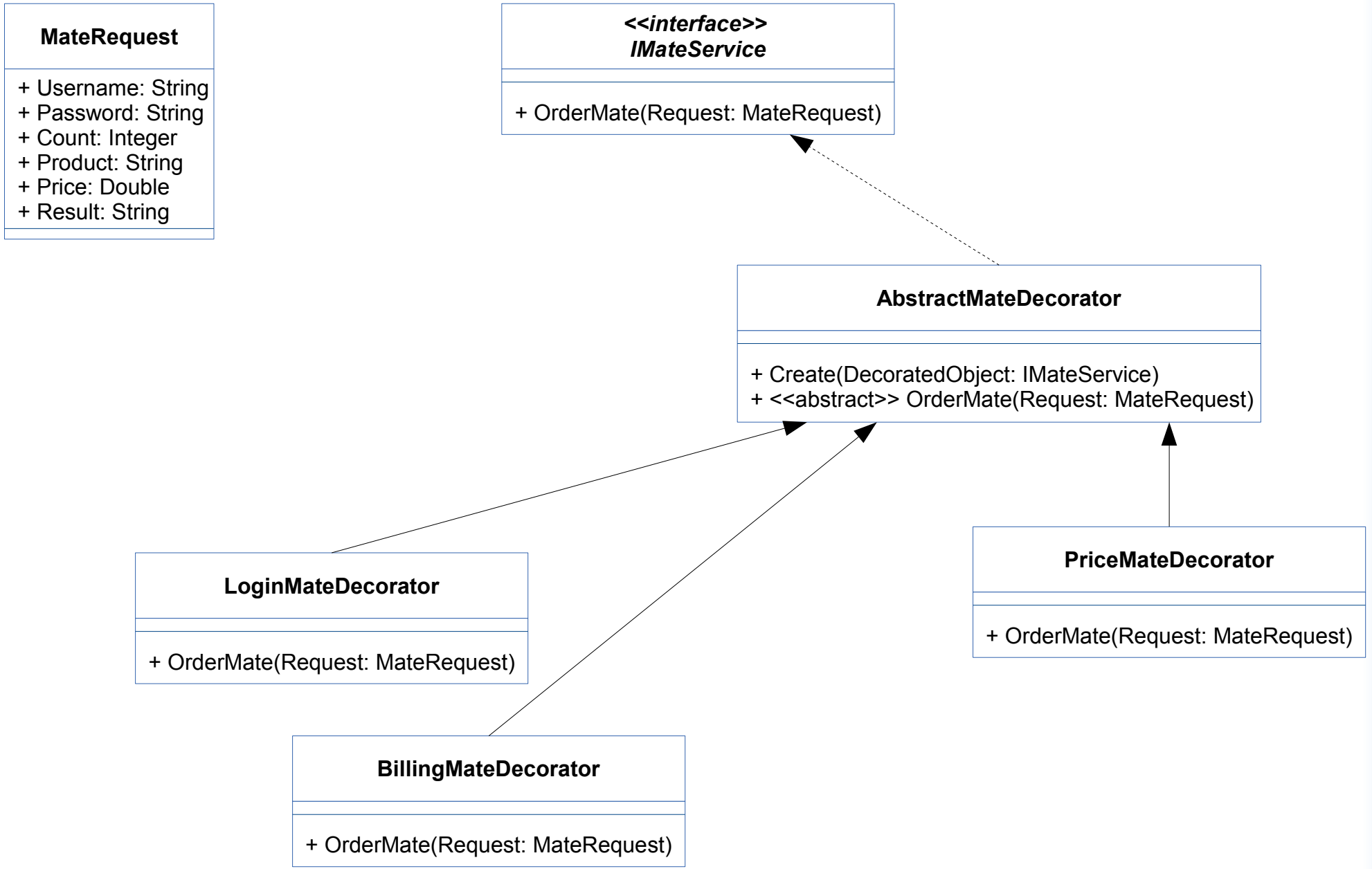

MateRequest
+ Username: String
+ Password: String
+ Count: Integer
+ Product: String
+ Price: Double
+ Result: String

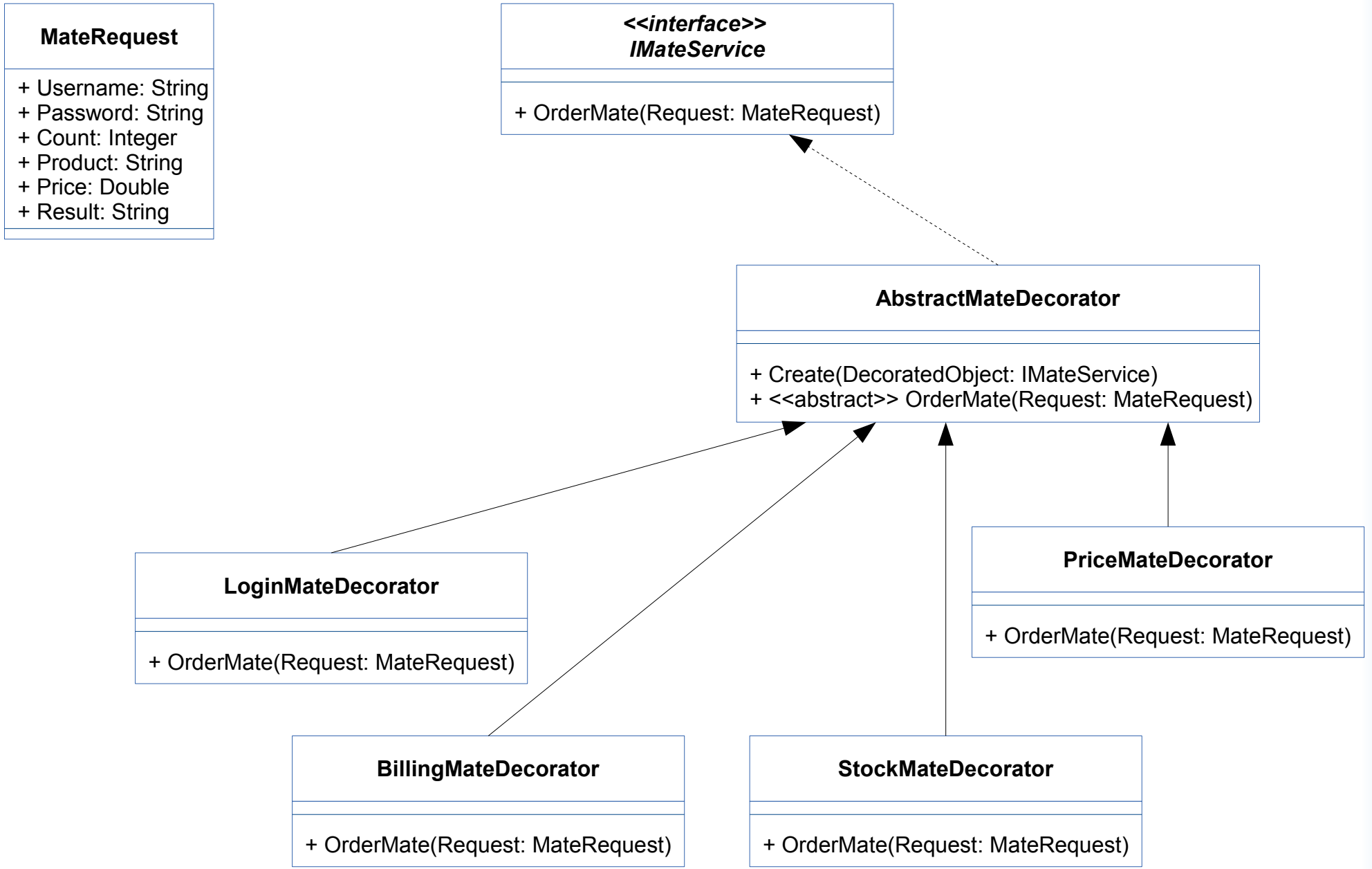


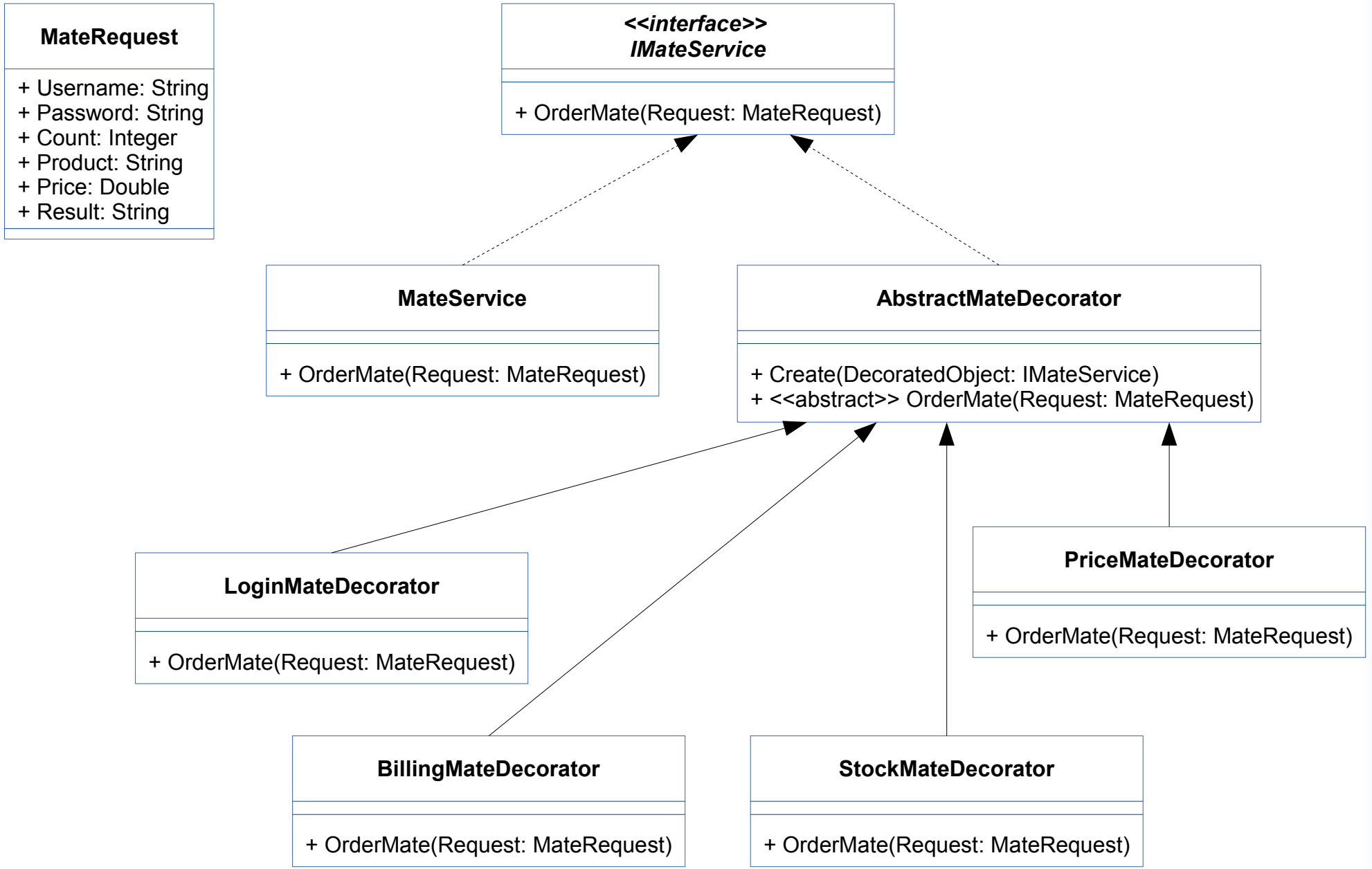

MateRequest
+ Username: String
+ Password: String
+ Count: Integer
+ Product: String
+ Price: Double
+ Result: String
<<interface>>
IMateService
+ OrderMate(Request: MateRequest)
AbstractMateDecorator
+ Create(DecoratedObject: IMateService)
+ <<abstract>> OrderMate(Request: MateRequest)



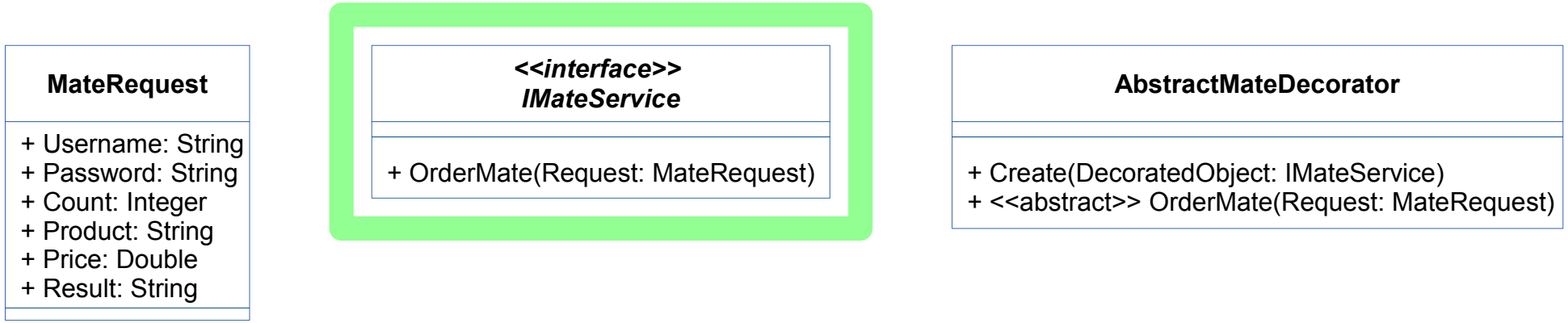

MateRequest
+ Username: String
+ Password: String
+ Count: Integer
+ Product: String
+ Price: Double
+ Result: String
<<interface>>
IMateService
+ OrderMate(Request: MateRequest)
AbstractMateDecorator
+ Create(DecoratedObject: IMateService)
+ <<abstract>> OrderMate(Request: MateRequest)
LoginMateDecorator
+ OrderMate(Request: MateRequest)
PriceMateDecorator
+ OrderMate(Request: MateRequest)



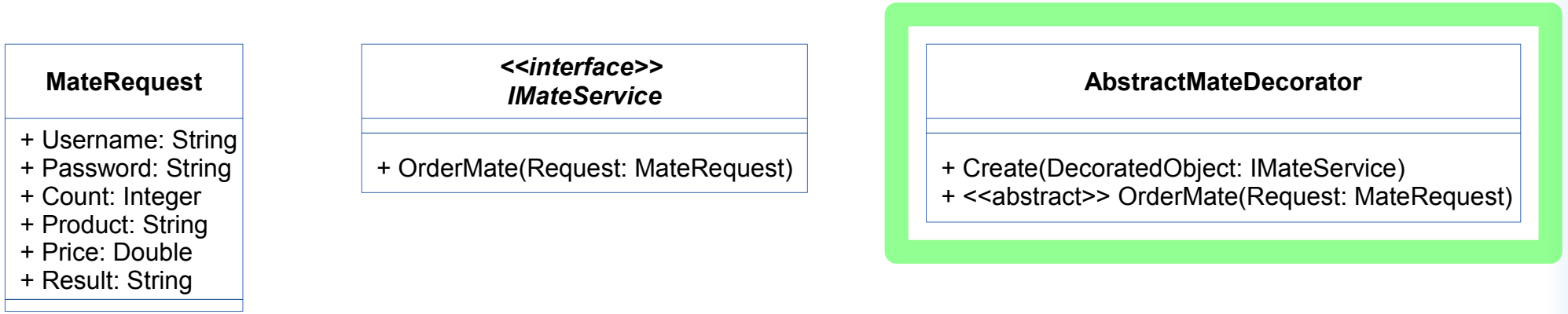




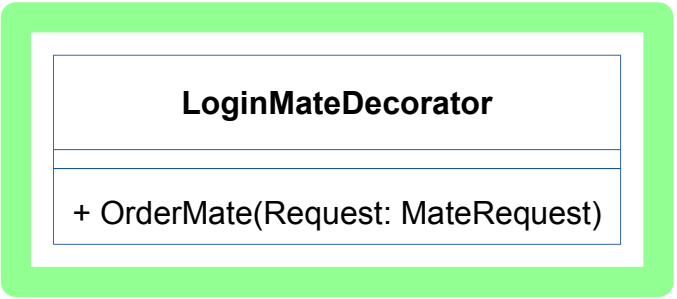

```
TMateRequest = class(TObject)
  public
    Username: String;
    Password: String;
    Count: Integer;
    Product: String;
    Price: Double;
    Result: String;
  public
    constructor Create(AUsername, APassword: String;
                       ACount: Integer; AProduct: String);
end;
```



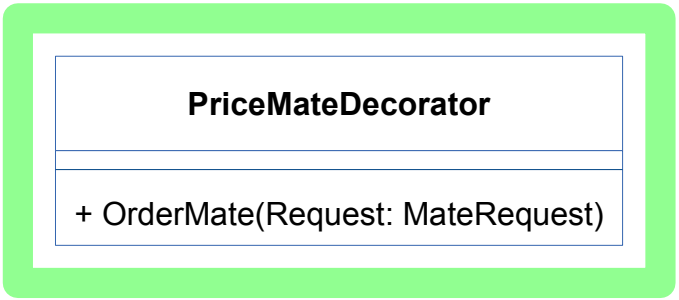

```
IMateService = interface
  procedure OrderMate(Request: TMateRequest);
end;
```



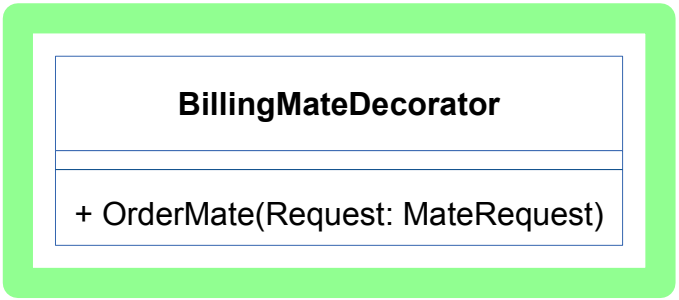

```
TAbstractMateDecorator = class(IMateService)
    protected
      DecoratedObject: IMateService;
    public
      constructor Create(ADecoratedObject: IMateService);
    public
      procedure OrderMate(Request: TMateRequest); virtual abstract;
  end;

(* ... *)

constructor TAbstractMateDecorator.Create(
                                ADecoratedObject: IMateService);
begin
  inherited Create;
  DecoratedObject := ADecoratedObject;
end;
```



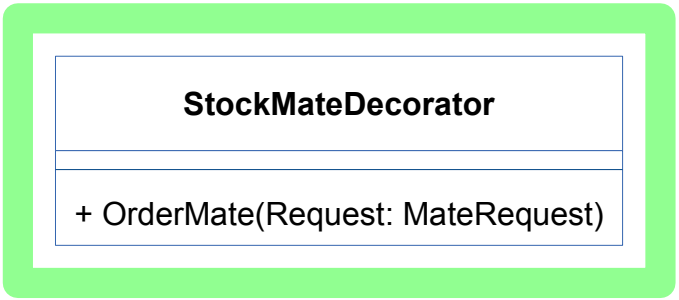

```
TLoginMateDecorator = class(TAbstractMateDecorator)
  public
    procedure OrderMate(Request: TMateRequest); override;
end;

(* ... *)

procedure TLoginMateDecorator.OrderMate(Request: TMateRequest);
begin
  if TUserService.Authenticate(Request.Username,
                               Request.Password) then
    Self.DecoratedObject.OrderMate(Request)
  else
    Request.Result := 'login failed';
end;
```




```
TPriceMateDecorator = class(TAbstractMateDecorator)
  public
    procedure OrderMate(Request: TMateRequest); override;
end;

(* ... *)

procedure TPriceMateDecorator.OrderMate(Request: TMateRequest);
var
  SinglePrice: Double;
begin
  SinglePrice := TDbService.GetPrice(Request.Product);
  Request.Price := Request.Count * SinglePrice;
  if TDbService.HasDiscount(Request.Username) then
    Request.Price := Request.Price * 0.9;
  Self.DecoratedObject.OrderMate(Request);
end;
```




```
TBillingMateDecorator = class(TAbstractMateDecorator)
  public
    procedure OrderMate(Request: TMateRequest); override;
end;

(* ... *)

procedure TBillingMateDecorator.OrderMate(Request: TMateRequest);
var
  Contingent: Double;
begin
  Contingent := TDbService.GetUserContingent(Request.Username);
  Contingent := Contingent - Request.Price;
  if Contingent >= 0 then begin
    Self.DecoratedObject.OrderMate(Request);
    if Request.Result = 'OK' then
      TDbService.SetUserContingent(Request.Username, Contingent);
  end else
    Request.Result := 'not enough credits';
end;
```




```
TStockMateDecorator = class(TAbstractMateDecorator)
  public
    procedure OrderMate(Request: TMateRequest); override;
end;

(* ... *)

procedure TStockMateDecorator.OrderMate(Request: TMateRequest);
var
  Stock: Integer;
begin
  Stock := TDbService.GetStock(Request.Product);
  Stock := Stock - Request.Count;
  if Stock >= 0 then begin
    Self.DecoratedObject.OrderMate(Request);
    if Request.Result = 'OK' then
      TDbService.SetStock(Request.Product, Stock);
  end else
    Request.Result := 'not enough in stock';
end;
```


```
TMateService = class(IMateService)
  public
    procedure OrderMate(Request: TMateRequest);
end;

(* ... *)

procedure TMateService.OrderMate(Request: TMateRequest);
var
  Address, Mailtext: String;
begin
  Address := TDbService.GetUserAdress(Request.Username);
  Mailtext := 'Liebes Versand-Team,' + LineEnding
            + 'schicke ' + IntToStr(Request.Count) + 'x '
            + Request.Product + ' an:' + LineEnding
            + '   ' + Request.Username + LineEnding
            + '   ' + Address;
  TMailService.SendMail('ship@mate24.de', 'Order', Mailtext);
  Request.Result := 'OK';
end;
```

Instanzierung

```
var
  Service: TMateService;
  StockService: TStockMateDecorator;
  BillingService: TBillingMateDecorator;
  PriceService: TPriceMateDecorator;
  LoginService: TLoginMateDecorator;
  Request: TMateRequest;
```

Instanzierung

```
Service        := TMateService.Create;
StockService   := TStockMateDecorator.Create(Service);
BillingService := TBillingMateDecorator.Create(StockService);
PriceService   := TPriceMateDecorator.Create(BillingService);
LoginService   := TLoginMateDecorator.Create(PriceService);
```

Beispielaufruf

```
Service        := TMateService.Create;
StockService   := TStockMateDecorator.Create(Service);
BillingService := TBillingMateDecorator.Create(StockService);
PriceService   := TPriceMateDecorator.Create(BillingService);
LoginService   := TLoginMateDecorator.Create(PriceService);
```

```
Request := TMateRequest.Create('seven.of.nine', 'resistance',
                               3, 'Mate Classic');
Service.OrderMate(Request);
WriteLn;
WriteLn('RESULT: ', Request.Result);
```

Beispielaufruf

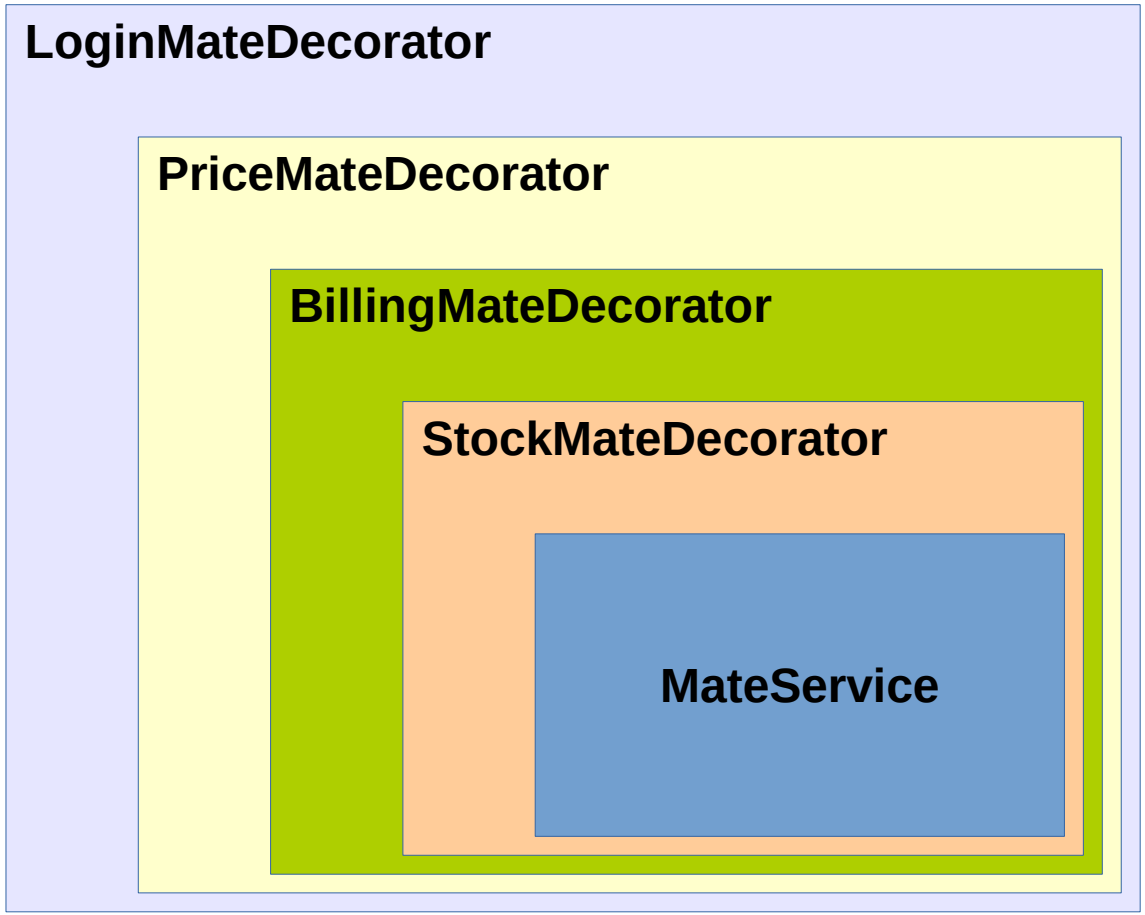
```
Service        := TMateService.Create;
StockService   := TStockMateDecorator.Create(Service);
BillingService := TBillingMateDecorator.Create(StockService);
PriceService   := TPriceMateDecorator.Create(BillingService);
LoginService   := TLoginMateDecorator.Create(PriceService);
```

```
Request := TMateRequest.Create('seven.of.nine', 'resistance',
                               3, 'Mate Classic');
Service.OrderMate(Request);
WriteLn;
WriteLn('RESULT: ', Request.Result);
```

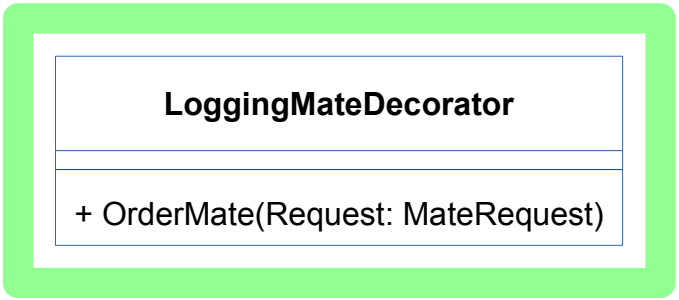
Ausgabe:

```
To: ship@mate24.de
Subject: Order
Liebes Versand-Team,
schicke 3x Mate Classic an:
  seven.of.nine
  Astrometrics, NCC-74656

RESULT: OK
```

flexibler Aufbau

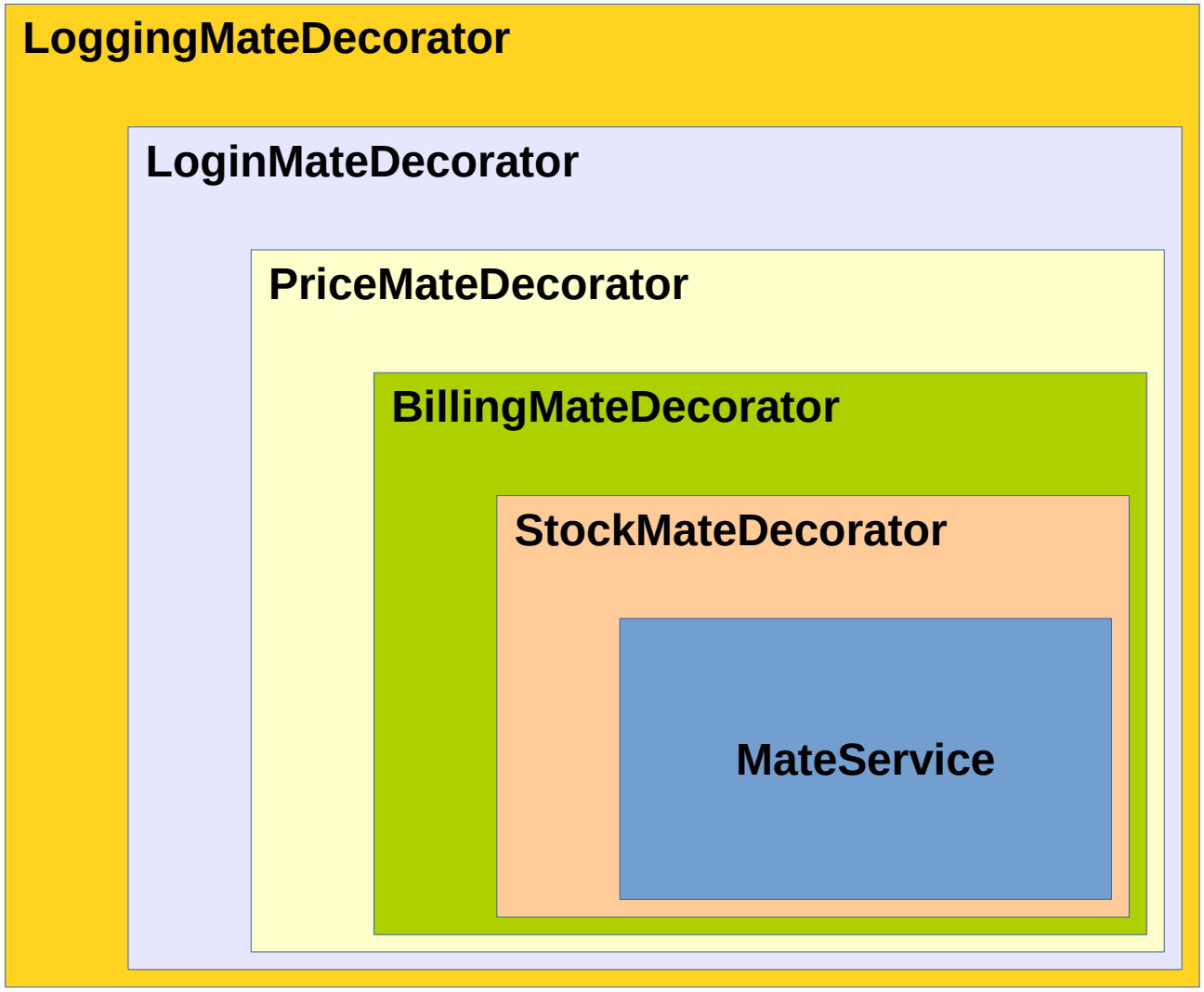


flexibler Aufbau




```
TLoggingMateDecorator = class(TAbstractMateDecorator)
  public
    procedure OrderMate(Request: TMateRequest); override;
end;

(* ... *)

procedure TLoggingMateDecorator.OrderMate(Request: TMateRequest);
begin
  WriteLn(Format('LOG: call OrderMate(%s, %s, %d, %s)',
                 [Request.Username, Request.Password,
                  Request.Count, Request.Product]));
  Self.DecoratedObject.OrderMate(Request);
end;
```

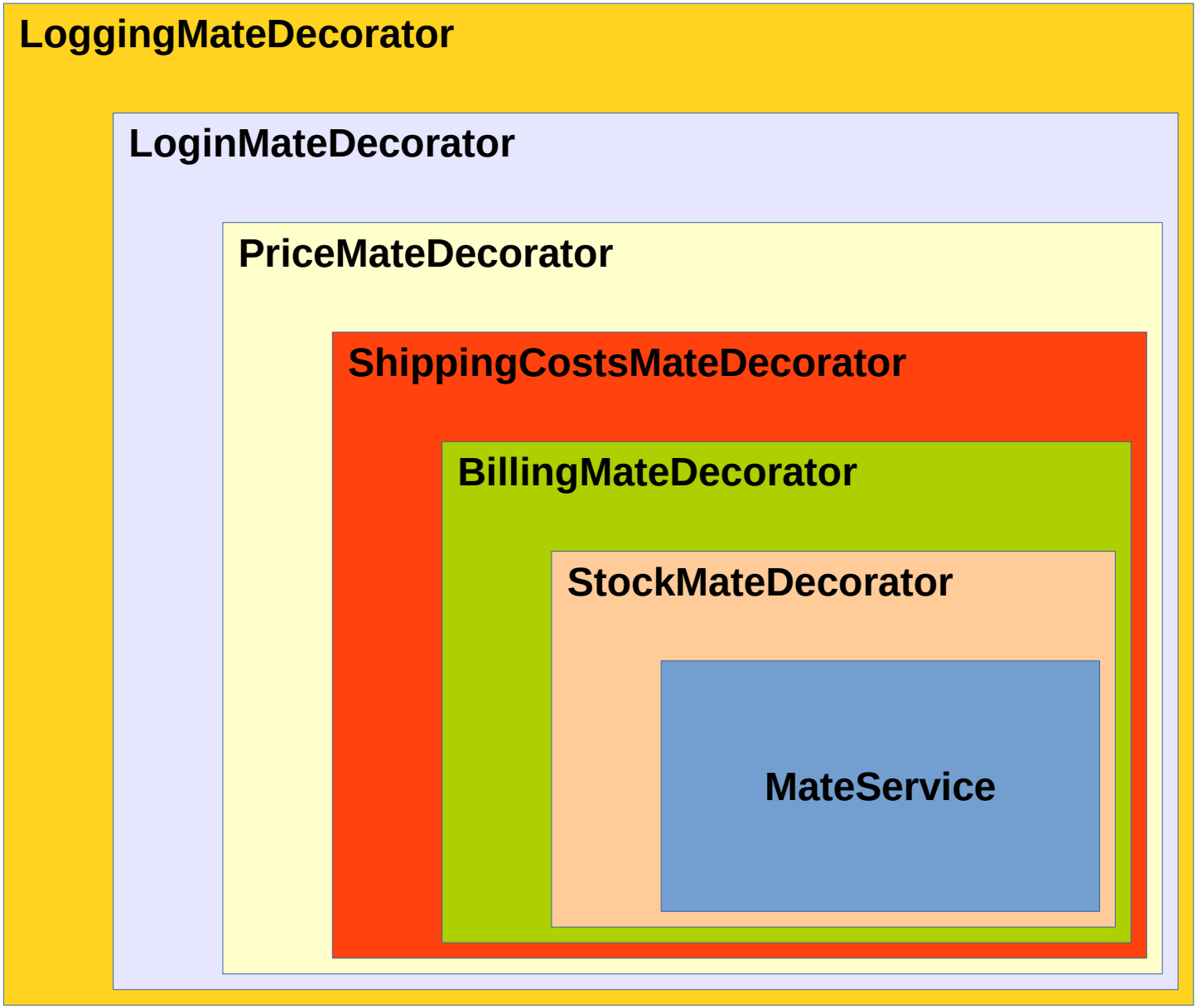
flexibler Aufbau



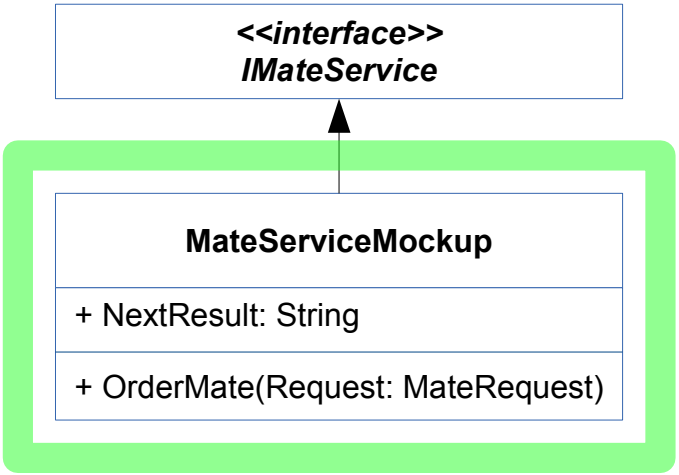
flexibler Aufbau


```
TShippingCostMateDecorator = class(TAbstractMateDecorator)
  public
    procedure OrderMate(Request: TMateRequest); override;
end;

(* ... *)

procedure TShippingCostMateDecorator.OrderMate(Request: TMateRequest);
begin
  if Request.Price < 25.0 then
    Request.Price := Request.Price + 2.5;
  Self.DecoratedObject.OrderMate(Request);
end;
```

flexibler Aufbau

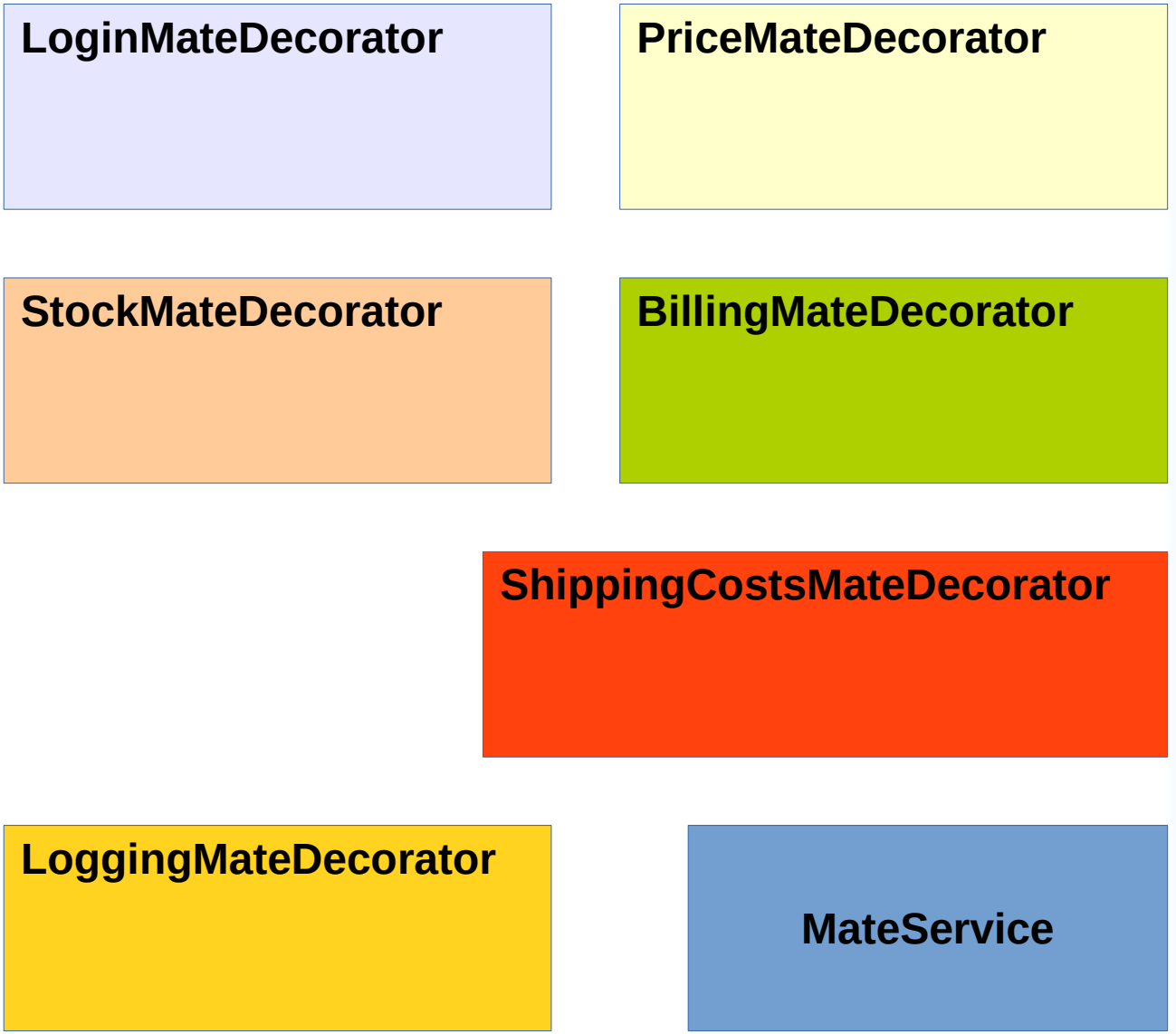


einfache Testbarkeit

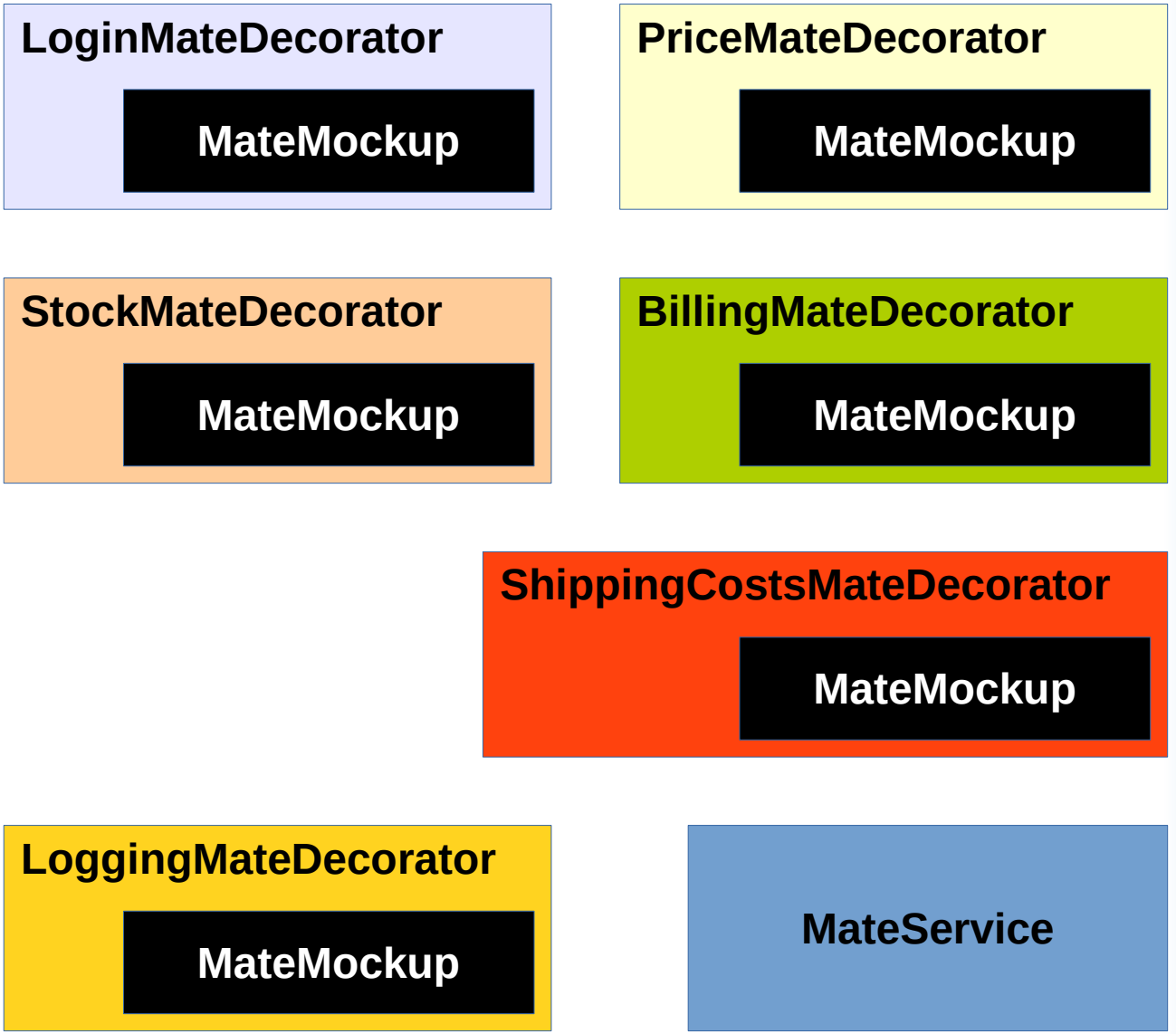



```
TMateServiceMockup = class(IMateService)
  public
    NextResult: String;
  public
    procedure OrderMate(Request: TMateRequest);
end;

(* ... *)

procedure TMateServiceMockup.OrderMate(Request: TMateRequest);
begin
  Request.Result := Self.NextResult;
end;
```

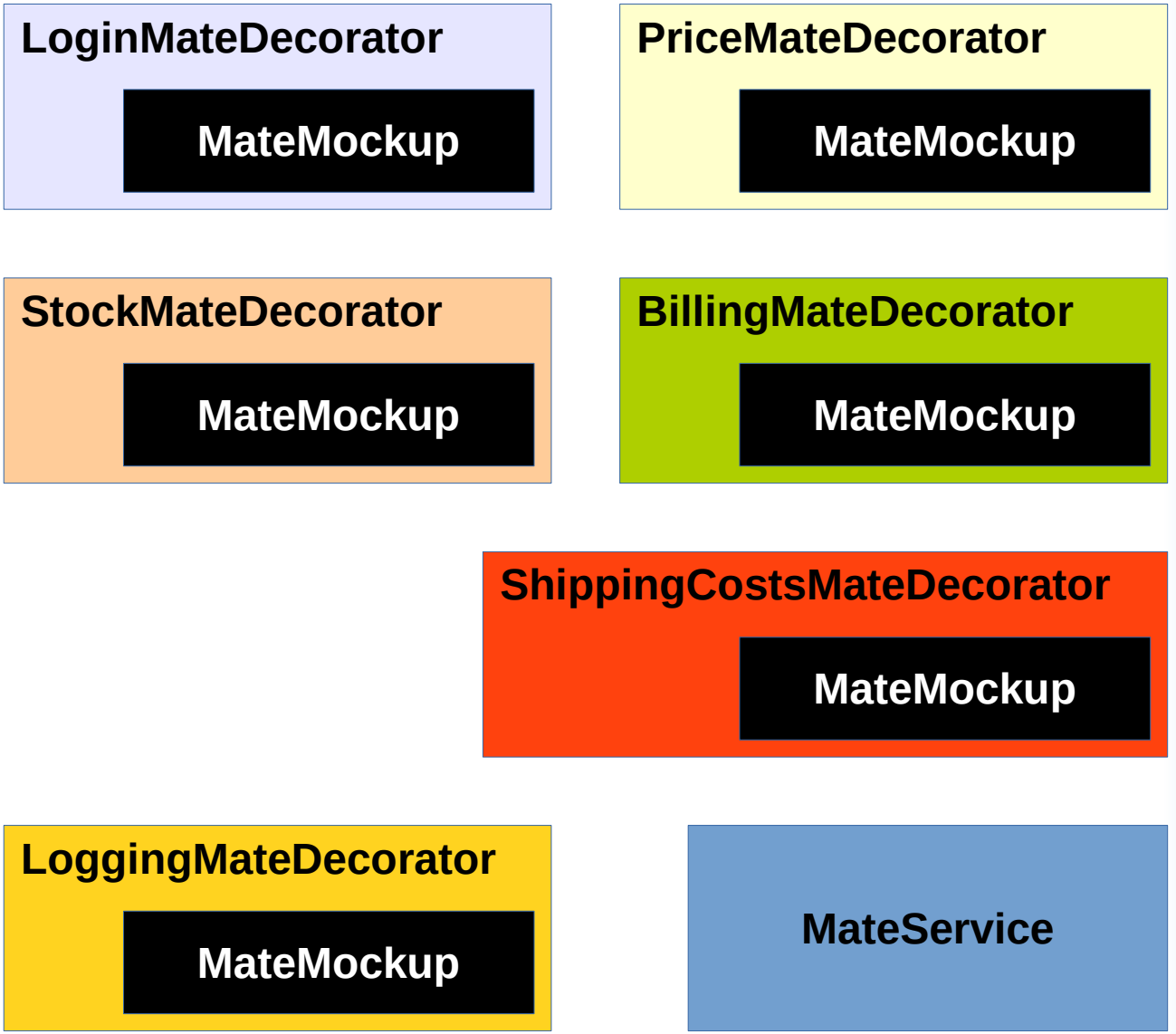
einfache Testbarkeit



einfache Testbarkeit



einfache Testbarkeit



- Fallstudie
  - Lösungsansatz: Klassische Vererbung
  - Lösungsansatz: Objektattribute
  - Lösungsansatz: Decorator
- Definition
- Refaktorisierungs-Beispiel
- **Ausblick**

Vorteile

Nachteile

Vorteile

- Vermeidung von Codeduplizierung

Nachteile

Vorteile

- Vermeidung von Codeduplizierung
- Aufteilung von großen Klassen in fachspezifische Teile

Nachteile

Vorteile

- Vermeidung von Codeduplizierung
- Aufteilung von großen Klassen in fachspezifische Teile
- verbesserte Testbarkeit / Tests von Teilbereichen möglich

Nachteile

Vorteile

- Vermeidung von Codeduplizierung
- Aufteilung von großen Klassen in fachspezifische Teile
- verbesserte Testbarkeit / Tests von Teilbereichen möglich
- flexible Reihenfolge

Nachteile

Vorteile

- Vermeidung von Codeduplizierung
- Aufteilung von großen Klassen in fachspezifische Teile
- verbesserte Testbarkeit / Tests von Teilbereichen möglich
- flexible Reihenfolge
- Reihenfolge während der Laufzeit änderbar

Nachteile

Vorteile

- Vermeidung von Codeduplizierung
- Aufteilung von großen Klassen in fachspezifische Teile
- verbesserte Testbarkeit / Tests von Teilbereichen möglich
- flexible Reihenfolge
- Reihenfolge während der Laufzeit änderbar
- transparent für den Aufrufer

Nachteile

Vorteile

- Vermeidung von Codeduplizierung
- Aufteilung von großen Klassen in fachspezifische Teile
- verbesserte Testbarkeit / Tests von Teilbereichen möglich
- flexible Reihenfolge
- Reihenfolge während der Laufzeit änderbar
- transparent für den Aufrufer
- Vermeidung von komplexen Vererbungshierarchien

Nachteile

Vorteile

- Vermeidung von Codeduplizierung
- Aufteilung von großen Klassen in fachspezifische Teile
- verbesserte Testbarkeit / Tests von Teilbereichen möglich
- flexible Reihenfolge
- Reihenfolge während der Laufzeit änderbar
- transparent für den Aufrufer
- Vermeidung von komplexen Vererbungshierarchien

Nachteile

- mehr Quellcode

Vorteile

- Vermeidung von Codeduplizierung
- Aufteilung von großen Klassen in fachspezifische Teile
- verbesserte Testbarkeit / Tests von Teilbereichen möglich
- flexible Reihenfolge
- Reihenfolge während der Laufzeit änderbar
- transparent für den Aufrufer
- Vermeidung von komplexen Vererbungshierarchien

Nachteile

- mehr Quellcode
- Prüfungen auf Typgleichheit können zu falschen Ergebnissen führen

Einsatzgebiete

Einsatzgebiete

- **Erweiterungen von Datenausgaben**

Einsatzgebiete

- Erweiterungen von Datenausgaben
- Codetrennung

Einsatzgebiete

- Erweiterungen von Datenausgaben
- Codetrennung
- Filter

Einsatzgebiete

- Erweiterungen von Datenausgaben
- Codetrennung
- Filter
- Eingabe – Validierung

Einsatzgebiete

- Erweiterungen von Datenausgaben
- Codetrennung
- Filter
- Eingabe – Validierung
- Funktionserweiterung

Einsatzgebiete

- Erweiterungen von Datenausgaben
- Codetrennung
- Filter
- Eingabe – Validierung
- Funktionserweiterung
- ...

Fragen