



Odissee
DE CO-HOGESCHOOL

Interfaces



Jens Baetens



Probleem



Situatie

- ▣ Eenvoudige applicatie waarbij er op 3 manieren moet kunnen betaald worden
- ▣ Elke betaling heeft zijn eigen afhandelscherm
 - Hou zou je de betaalbutton implementeren?



Mogelijke oplossing

```
if(visaRadioButton.IsChecked == true)
{
    Visa visaPayment = new Visa();
    visaPayment.ShowDialog();
    if(visaPayment.PaymentSucceeded)
    {
        MessageBox.Show("Visa Payment succeeded");
    }
    else
    {
        MessageBox.Show("Visa Payment failed");
    }
} else if (paypalRadioButton.IsChecked == true)
{
    Paypal paypalPayment = new Paypal();
    paypalPayment.ShowDialog();
    if (paypalPayment.PaymentSucceeded)
    {
        MessageBox.Show("Paypal Payment succeeded");
    }
    else
    {
        MessageBox.Show("Paypal Payment failed");
    }
} else if (bancontactRadioButton.IsChecked == true)
{
    Bancontact bancontactPayment = new Bancontact();
    bancontactPayment.ShowDialog();
    if (bancontactPayment.PaymentSucceeded)
    {
        MessageBox.Show("Bancontact Payment succeeded");
    }
    else
    {
        MessageBox.Show("Bancontact Payment failed");
    }
}
```



Wat valt je hierbij op?



Wat valt je hierbij op?

- ▣ Veel gelijkaardige code
- ▣ Elke betaalklasse bevat
 - methode om het scherm te tonen
 - Property om te valideren dat de betaling geslaagd is
 - Gelijkaardige boodschappen om te tonen
- ▣ De publieke methodes en properties worden ook wel de interface genoemd



Interface

Wat is een interface?

- Geen GUI (graphical user interface)
 - ▬ Wel gelijkaardige functie
 - ▬ GUI is een interface tussen applicatie en gebruiker
 - Gebruiker gebruikt de applicatie door middel van de interface: buttons, textboxes, listboxes, ...
- Een interface van een klasse zijn de methodes en properties die door een andere klasse gebruikt kunnen worden

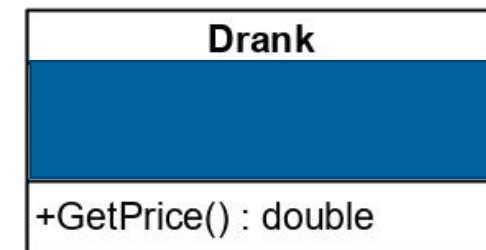
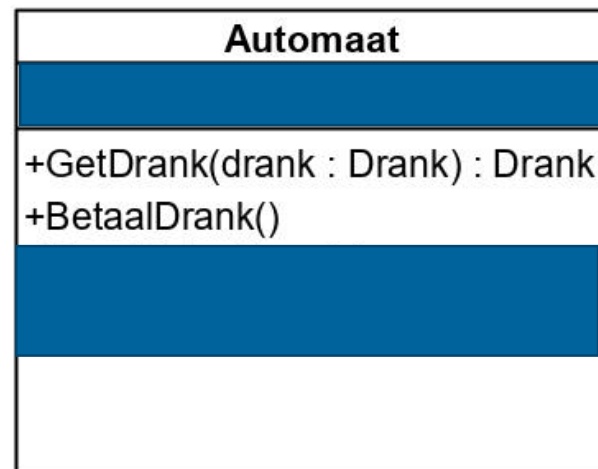


Automaat
-dranken : List<Drank>
+GetDrank(drank : Drank) : Drank
+BetaalDrank()
-MovePicker() : void
-ReturnChange() : double

Drank
-prijs : double
-ingredienten : List<string>
+GetPrice() : double

Wat is een interface?

- ▣ Geen GUI (graphical user interface)
 - ▬ Wel gelijkaardige functie
 - ▬ GUI is een interface tussen applicatie en gebruiker
 - Gebruiker gebruikt de applicatie door middel van de interface: buttons, textboxes, listboxes, ...
- ▣ Een interface van een klasse zijn de methodes en properties die door een andere klasse gebruikt kunnen worden





Definitie

- ▣ Een beschrijving van een klasse zonder expliciete implementatie
- ▣ Vertelt “de gebruiker” welke properties in een methode, klasse aanwezig zijn
 - ▣ Gebruiker is hier de klasse die de methodes oproept

Interface voorbeeld/demo

- ▣ Keyword Interface
- ▣ Geprefixt met hoofdletter i
- ▣ Interface kan public en internal zijn
- ▣ Alle methodes en properties in een interface zijn publiek

```
6 references
interface IPayable
{
    4 references
    void OpenPaymentScreen();
    4 references
    bool PaymentSucceeded { get; }
    4 references
    string PaymentSucceededMessage { get; }
    4 references
    string PaymentFailedMessage { get; }
}
```

Interface implementeren

- ▣ Interface alleen is waardeloos
 - Een klasse moet hem implementeren
- ▣ Een interface implementeren
 - = verplichten bepaalde methoden te voorzien
- ▣ Wordt toegevoegd achter het dubbelpunt
 - Gelijkaardig aan overerving
- ▣ Alle public properties en methoden moeten geïmplementeerd worden

```
4 references
public partial class Paypal : Window, IPayable
{
    private bool _paypalPaymentSucceeded = false;

    2 references
    public Paypal()
    {
        InitializeComponent();
    }

    //interface IPayable
    2 references
    public void OpenPaymentScreen()
    {
        ShowDialog();
    }

    2 references
    public bool PaymentSucceeded => _paypalPaymentSucceeded;

    2 references
    public string PaymentSucceededMessage
    {
        get => "Payment with paypal succeeded";
    }

    2 references
    public string PaymentFailedMessage
    {
        get => "Payment with paypal failed";
    }
    // end interface IPayable

    1 reference
    private void payButton_Click(object sender, RoutedEventArgs e)
    {
    }
}
```

Interface gebruiken

- ▣ Variabelen die van een interface type zijn
 - Kan van elke klasse die het implementeert zijn
- ▣ Een interface is **nooit** een object

```
IPayable payWindow = null;
if (visaRadioButton.IsChecked == true)
{
    payWindow = new Visa();
}
else if (paypalRadioButton.IsChecked == true)
{
    payWindow = new Paypal();
}
else if (bancontactRadioButton.IsChecked == true)
{
    payWindow = new Bancontact();
}

if (payWindow != null)
{
    payWindow.OpenPaymentScreen();
    if (payWindow.PaymentSucceeded)
    {
        MessageBox.Show(payWindow.PaymentSucceededMessage);
    }
    else
    {
        MessageBox.Show(payWindow.PaymentFailedMessage);
    }
}
```

Oefening

- ▣ Maak een console project aan
- ▣ Maak een interface ISound met 1 methode MakeNoise()
- ▣ Maak drie klassen aan die deze interface implementeren
 - ▬ Human die in de console “speak” print
 - ▬ Dog die in de console “woof” print
 - ▬ Cat die in de console “miauw” print
- ▣ Maak een lijst met een object van elk van bovenstaande klassen
 - ▬ Voer de MakeNoise() functie uit op elk object
- ▣ Wat is de output?

Oplossing

```
List<ISound> sounds = new List<ISound>();
sounds.Add(new Human());
sounds.Add(new Dog());
sounds.Add(new Cat());

foreach (var s in sounds)
{
    s.MakeNoise();
}
```

```
5 references
internal interface ISound
{
    4 references
    void MakeNoise();
}
```

```
1 reference
internal class Human: ISound
{
    2 references
    public void MakeNoise()
    {
        Console.WriteLine("speak");
    }
}
```

```
1 reference
internal class Cat: ISound
{
    2 references
    public void MakeNoise()
    {
        Console.WriteLine("Miauw");
    }
}
```

```
1 reference
internal class Dog: ISound
{
    2 references
    public void MakeNoise()
    {
        Console.WriteLine("Woof");
    }
}
```


Nut van een interface

- ▣ Interface als ontwerpbeschrijving
 - ▬ Afspraken tussen developers
 - Zo kunnen ze onafhankelijk werken zonder de implementatie nodig te hebben
- ▣ Handig in Unit Testing voor Mocks en Stubs te maken

Nut van een interface

▣ Interface als onderlinge aansluiting

- Afdwingen van bepaalde elementen in een klasse
- Zorgt ervoor dat componenten gemakkelijk aangepast, gewisseld en samengevoegd kunnen worden

Reeds bestaande voorbeelden

- ▣ Interfaces worden heel veel gebruikt in allerlei programmeertalen
 - IList
 - IDictionary
 - IEnumerable
 - IComparable
 - IFormatable
- ▣ Met deze interfaces kunnen we eigen implementaties die vlot in het C# .NetFramework gebruikt kunnen worden

Eigenschappen interfaces

▣ 1 Klasse kan

- ▬ Van 1 klasse overerven
- ▬ Meerdere interfaces kunnen geïmplementeerd worden
- ▬ <https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1?view=netframework-4.8>

▣ Bevat enkel publieke methodes en properties

- ▬ Zonder implementatie!

Dependencies

▣ Dependency of afhankelijkheid

- Een klasse die een andere klasse gebruikt
- Maakt testen moeilijker (Moeilijk om afzonderlijk te testen)
- Maakt moeilijker om afzonderlijke delen te maken

2 references

```
public partial class MainWindow : Window
{
    private Calculator calculator = new Calculator();

    0 references
    public MainWindow()
    {
        InitializeComponent();
    }
}
```

Mainwindow heeft een afhankelijkheid van Calculator

Mainwindow werkt dus niet als Calculator niet goed werkt

Dependency Injection

- ▣ Dynamisch injecteren / toevoegen van afhankelijkheden
 - Typisch via de constructor
- ▣ Er wordt aangegeven met behulp van een **interface** welk type object er verwacht wordt maar exacte klasse maar gekend na compilatie.

```
public partial class MainWindow : Window
{
    private ICalculator _calculator;

    0 references
    public MainWindow(ICalculator calculator)
    {
        _calculator = calculator;
        InitializeComponent();
    }
}
```