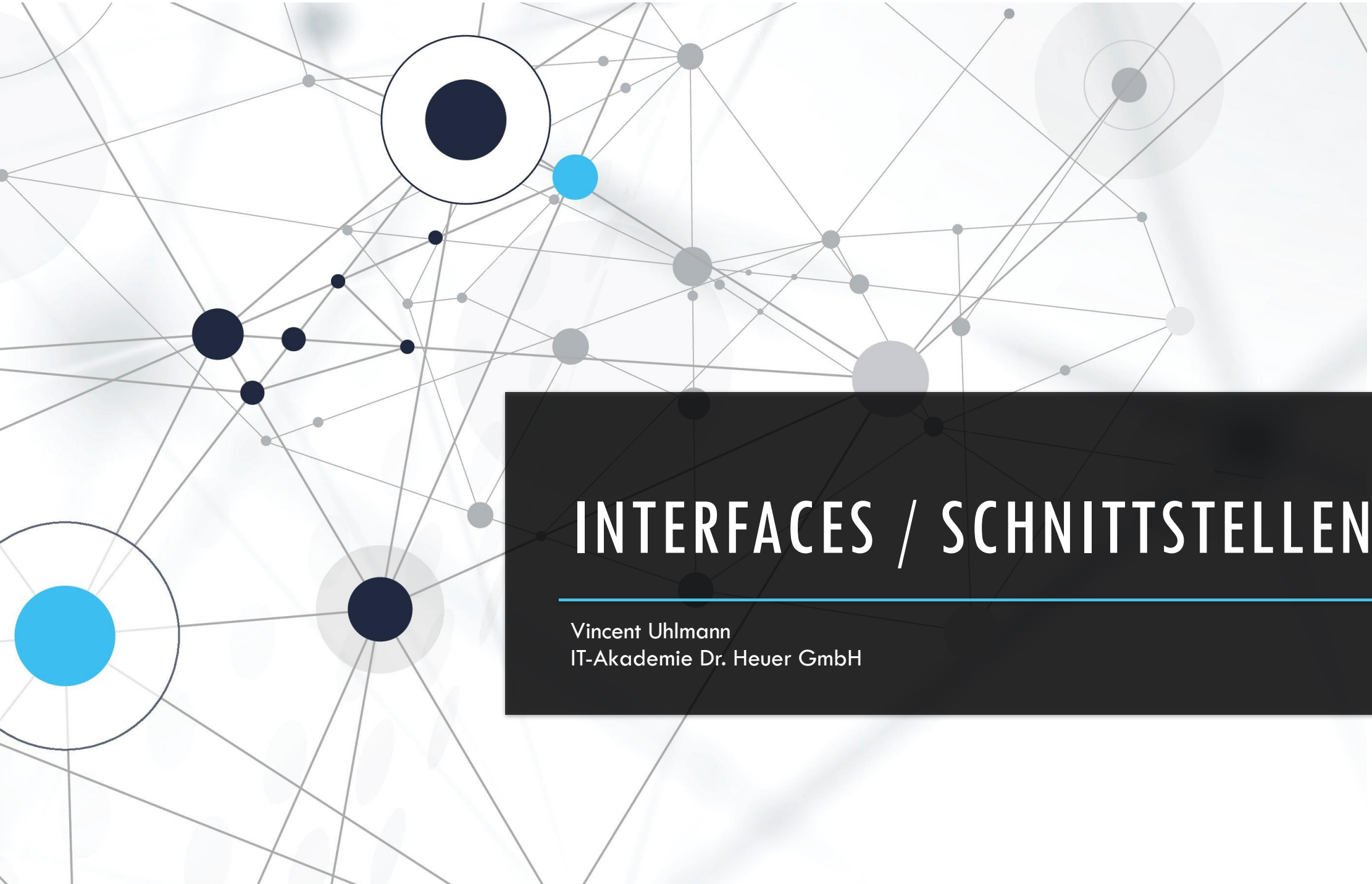


INTERFACES / SCHNITTSTELLEN

Vincent Uhlmann
IT-Akademie Dr. Heuer GmbH

INTERFACES / SCHNITTSTELLEN

- Klassen können in C# nur von genau einer Klasse erben
- Keine Mehrfachverberung möglich
- Abhilfe durch Schnittstellen (Interfaces)

- Schnittstellen ähneln abstrakten Klassen
- Definieren Verhaltensweisen, ohne Implementierung
- Bieten eine Vertragsvereinbarung für implementierte Klassen

FUNKTIONALITÄT

- Können Methoden, Eigenschaften (Properties), Events und Indexer vorschreiben
- Keine Felder oder Konstruktoren
- Definieren eine Art Vertragsvereinbarung für Implementierungen

```
public interface ILogger
{
    string this[int index]
    {
        get;
        set;
    }

    event EventHandler LogLevelChanged;

    int LogLevel { get; }

    void LogInfo(string msg);
}
```

SYNTAX UND IMPLEMENTIERUNG

- Deklaration mit dem Schlüsselwort ‘interface’
- Der Name eines Interfaces beginnt mit einem großen I
  - Beispiel: ILogger
- Implementierung durch Angabe des Interface-Namens nach dem Klassen-Namen
- Interfaces können bei der Deklaration keine Zugriffsmodifizierer verwenden
- Bei der Implementierung müssen diese mit public deklartiert werden

```
public interface ILogger
{
    public void LogInfo(string msg); // Falsch
    void LogWarning(string msg); // Richtig
}
```

SYNTAX UND IMPLEMENTIERUNG

```
public interface ILogger
{
    void LogInfo(string msg);
    void LogWarning(string msg);
}
```

```
public class Logger : ILogger
{
    public void LogInfo(string msg)
    {
        Console.WriteLine("Info: " + msg);
    }

    public void LogWarning(string msg)
    {
        Console.WriteLine("Warning: " + msg);
    }
}
```

MEHRFACHE INTERFACE-IMPLEMENTIERUNG

- Eine Klasse kann mehrere Interfaces implementieren
- Erhöht die Flexibilität und Wiederverwendbarkeit des Codes
- Kombiniert verschiedene Verhaltensweisen in einer Klasse

```
public interface ILogger
{
    void LogInfo(string message);
}

public interface IDatabase
{
    void DatenbankVerbinden();
}
```

```
public class ProgramManager : ILogger, IDatabase
{
    public void LogInfo(string msg)
    {
        Console.WriteLine("Info: " + msg);
    }

    public void DatenbankVerbinden()
    {
        // Code um Datenbank zu verbinden
    }
}
```

TYPUMWANDLUNG

- Objekte implizit in jedes Interface casten, welches es implementiert

```
// Aufruf
ProgramManager pm = new ProgramManager();
ILogger logger = pm;
IDatabase database = pm;
```

```
public interface ILogger
{
    void LogInfo(string msg);
}

public interface IDatabase
{
    void DatenbankVerbinden();
}

public class ProgramManager : ILogger, IDatabase
{
    public void LogInfo(string msg)
    {
        Console.WriteLine("Info: " + msg);
    }

    public void DatenbankVerbinden()
    {
        // Code um Datenbank zu verbinden
    }
}
```

VIRTUELLE IMPLEMENTIERUNG

- Durch virtuelles Implementieren eines Interfaces in der Basisklasse kann die Standardimplementierung bereitgestellt werden
- Diese Methode kann in abgeleiteten Klassen überschrieben werden, um benutzerdefiniertes Verhalten hinzuzufügen

```
public interface ILogger
{
    void LogInfo(string message);
}
```

```
public class Logger : ILogger
{
    public virtual void LogInfo(string msg)
    {
        Console.WriteLine("Info: " + msg);
    }
}

public class NewLogger : Logger
{
    public override void LogInfo(string msg)
    {
        base.LogInfo("NewLogger " + msg);
    }
}
```

EXPLIZITE IMPLEMENTIERUNG

- Explizite Implementierung ermöglicht es, Methoden auszublenden und die Klarheit des Codes zu verbessern
- Die Methode ist dann nur sichtbar, wenn sie über eine Interface-Referenz aufgerufen wird

```
// Funktioniert
ILogger logger = new Logger();
logger.LogInfo("");

// Funktioniert nicht
Logger logger = new Logger();
logger.LogInfo("");
```

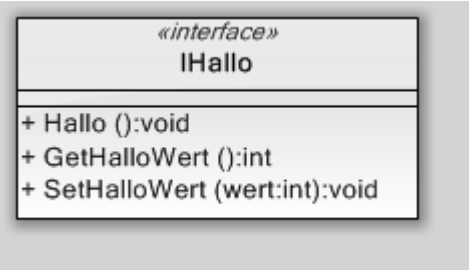
```
public interface ILogger
{
    void LogInfo(string msg);
}

public class Logger : ILogger
{
    void ILogger.LogInfo(string msg)
    {
        Console.WriteLine("Info: " + msg);
    }
}
```

GÄNGIGE INTERFACES

- IComparable
  - Ermöglicht das Vergleichen von Objekten eines bestimmten Typs
- IDisposable
  - Definiert Methoden zum manuellen Freigeben von Ressourcen
- ICloneable
  - Ermöglicht das Klonen von Objekten
- IEnumerable
  - Erlaubt eine einfache Iteration durch Auflistungen

INTERFACES IN DER UML

- Werden durch den Stereotyp <<interface>> gekennzeichnet
- Besitzen keine Attribute
- Enthalten nur Methoden
- Methodennamen wird immer ein + (public) vorangestellt



INTERFACES IN DER UML

- Klassen realisieren eine Schnittstelle
- Realisierungspfeil ähnlich zum Vererbungspfeil
- Jedoch mit gestrichelter Linie
- Klasse bindet Methoden der Schnittstelle ein