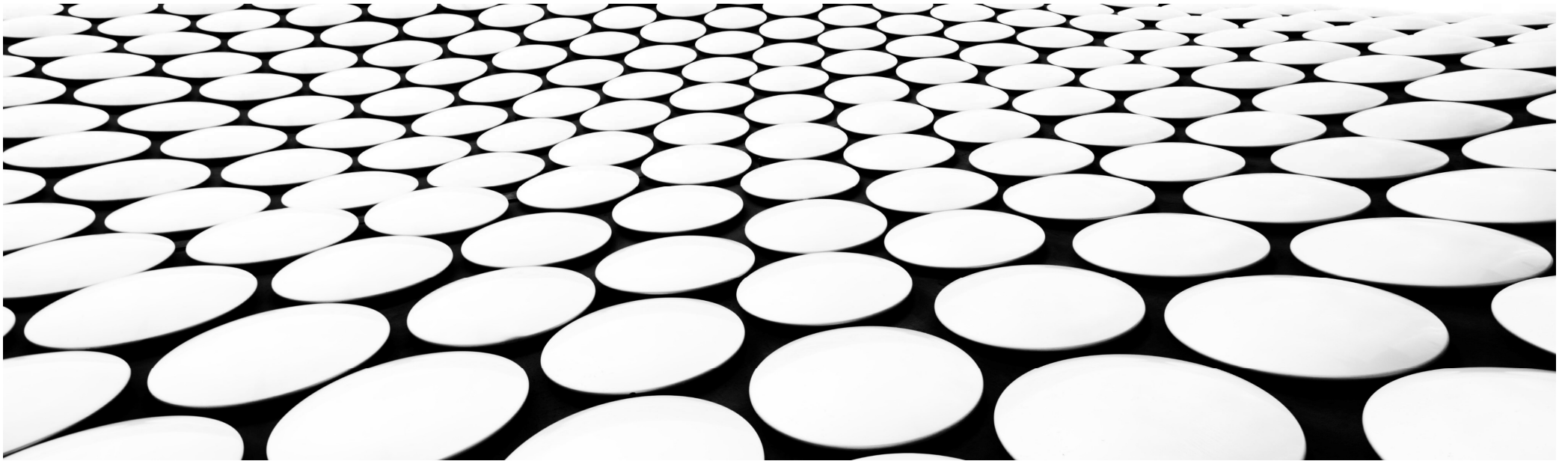

ABSTRAKTE KLASSEN & INTERFACES

BAUPLAN & VERTRAG



ABSTRAKTE KLASSEN & INTERFACES

Abstrakte Klassen

Interfaces




ABSTRAKTE KLASSEN



ABSTRAKTE KLASSEN

Eine abstrakte Klasse ist eine Klasse, von der keine Instanzen erzeugt werden können. Sie fasst Gemeinsamkeiten zusammen und gibt eine quasi „Schnittstelle“ vor, die von ihren abgeleiteten Klassen erfüllt werden muss.

Schlüsselwort zur Kennzeichnung von abstrakten Klassen



```
public abstract class Artikel {  
    ...  
}
```

ABSTRAKTE KLASSEN

Eine abstrakte Methode hat eine Signatur, aber keinen Methodenrumpf. Sie gibt vor, welche Funktionalität in Unterklassen implementiert werden müssen.

Klasse muss abstrakt sein, da sie eine abstrakte Methode deklariert

```
public abstract class Artikel {  
    ...  
    public abstract String getTwitterBeschreibung();  
}
```

Kennzeichnung der
abstrakten Methode


Abstrakte Methoden
definieren keinen
Methodenrumpf

ABSTRAKTE KLASSEN

In einer Unterklasse von „Artikel“
muss diese Methode
implementiert werden

Implementierung der abstrakten
Methode in der Unterklasse
Buch

```
public class Buch extends Artikel {  
    ...  
    public String getTwitterBeschreibung() {  
        return "Buch: '" + titel + "' von " + autor;  
    }  
}
```



ABSTRAKTE KLASSEN

- Das Implementieren einer geerbten abstrakten Methode in einer Unterklasse wird vom Compiler erzwungen
- Die einzige Möglichkeit, eine abstrakte Methode nicht in einer direkten Unterklasse zu implementieren, ist, sie dort wieder als „abstract“ zu kennzeichnen
- Das hat allerdings zur Folge, dass diese Unterklasse zu einer abstrakten Klasse wird und so die Verantwortung der Implementierung an weitere Unterklassen delegiert wird
- Es ist möglich, Variablen vom Typ einer abstrakten Klasse zu deklarieren
- Da aber keine Objekte von dieser abstrakten Klasse erzeugt werden können, muss der zugewiesene Wert der so deklarierten Variable zwingend ein Objekt einer abgeleiteten Klasse sein

ABSTRAKTE KLASSEN

```
public static void main(String[] args) {  
Artikel artikel = new Artikel();  
Artikel artikel;  
artikel = new Buch();  
System.out.println(  
    artikel.getTwitterBeschreibung());  
}
```

Instanziierung nicht erlaubt, da Artikel eine abstrakte Klasse ist

Eine Variable kann vom Typ einer abstrakten Klasse deklariert werden.

Zuweisung ist auf Grund von Zuweisungskompatibilität erlaubt

Für den Aufruf wird die Implementierung aus der Klasse Buch gewählt.



INTERFACES

SCHNITTSTELLEN



INTERFACES

- Ein Interface definiert eine Menge von Methoden, die von Klassen implementiert werden können
- Weil die Methoden erst in den Klassen realisiert werden, findet man in Interfaces nur Methoden ohne einen Rumpf

INTERFACES

- Schnittstellen werden in Java mit dem Schlüsselwort `interface` definiert. Die Syntax erinnert sehr an die einer Klasse, mit dem Unterschied, dass alle Methoden abstrakt sind.
- Die Methoden bestehen lediglich aus der Signatur und haben somit keinen durch geschweifte Klammern eingeschlossenen Rumpf
- Auf diese Weise können Sie spezifizieren, was eine implementierende Klasse können muss, ohne das „Wie?“ vorwegzunehmen

INTERFACES

Methoden müssen implementiert werden

```
public interface EineSchnittstelle {  
    public void eineMethode(int param);  
}
```

Name der Schnittstelle in der Definition und in der Implementierung

```
public class ImplementierendeKlasse implements EineSchnittstelle  
{  
    ...  
    public void eineMethode(int param){  
        ...  
    }  
    ...  
}
```

hier wird die Schnittstelle implementiert

es folgen ggfs. weitere Methoden

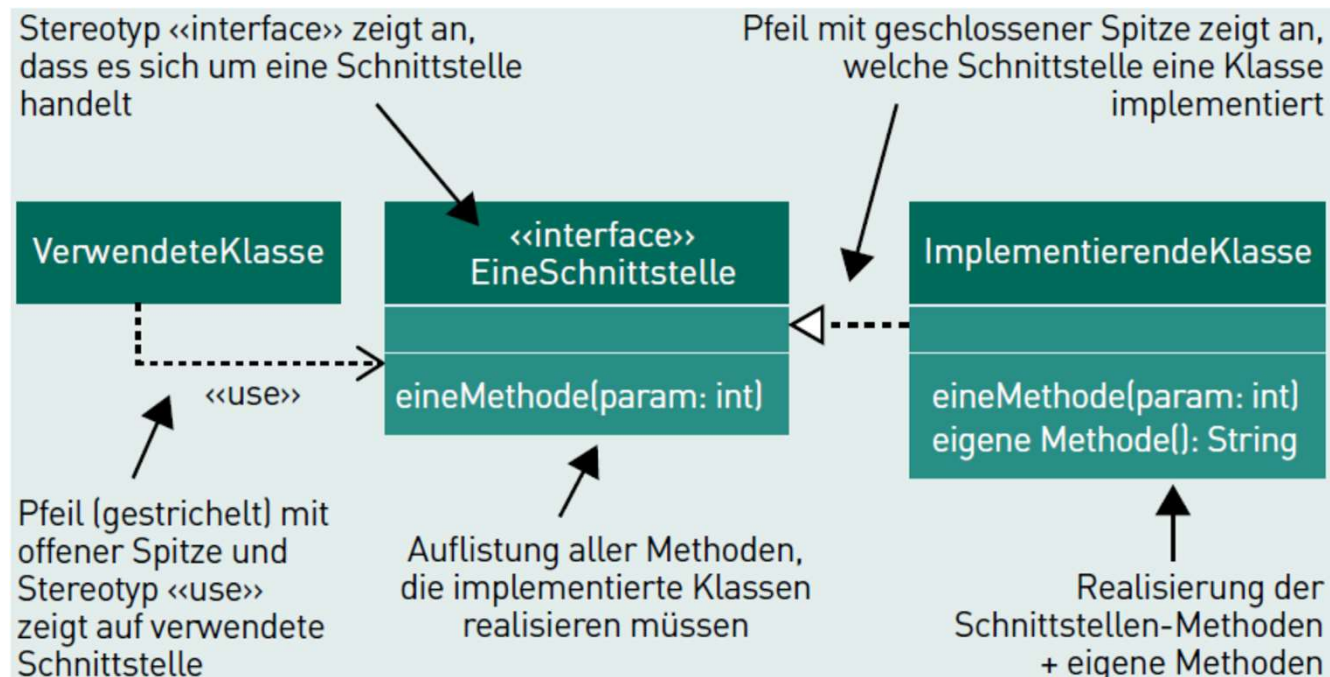
Rechteckiges Ausschneiden

The diagram illustrates the relationship between a Java interface and its implementation. It features two code blocks. The first block defines a public interface named 'EineSchnittstelle' with a single method 'eineMethode(int param)'. The second block defines a public class 'ImplementierendeKlasse' that implements 'EineSchnittstelle'. The class contains the implementation of 'eineMethode' and other methods indicated by ellipses. Annotations include a vertical box on the left stating 'Methoden müssen implementiert werden' with arrows pointing to the method signatures in both blocks. A horizontal arrow points from the interface name 'EineSchnittstelle' in the first block to the same name in the 'implements' clause of the second block, with the text 'Name der Schnittstelle in der Definition und in der Implementierung'. A bracket on the right side of the second block groups the implementation of 'eineMethode' and other methods, with the text 'hier wird die Schnittstelle implementiert'. Another arrow points to the ellipses in the second block with the text 'es folgen ggfs. weitere Methoden'. A small button labeled 'Rechteckiges Ausschneiden' is located near the top right of the second code block.

INTERFACES

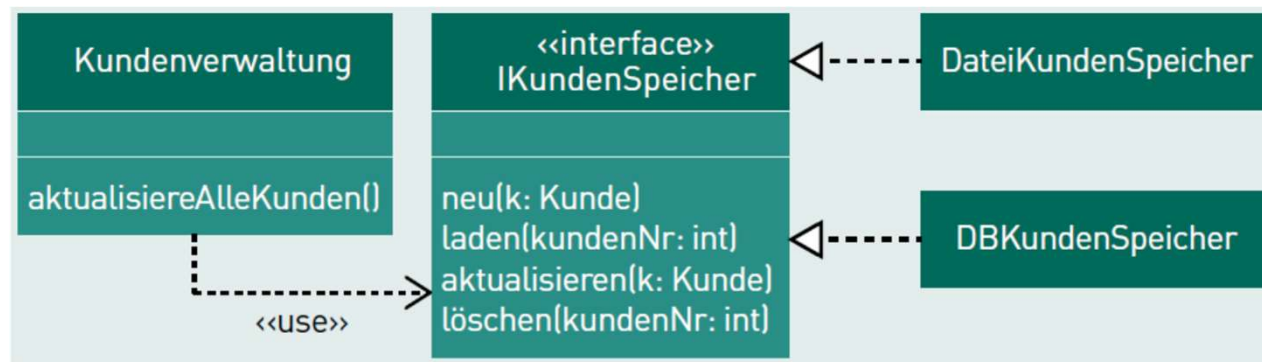
- In UML ähnelt die Notation von Schnittstellen der von Klassen
- Zur Unterscheidung wird das Stereotyp <<interface>> verwendet
- Mit einer use-Assoziation, dargestellt durch einen gestrichelten Pfeil mit offener Spitze und Stereotyp <<use>> wird gekennzeichnet, dass eine Klasse eine Schnittstelle verwendet. D.h., die Klasse ruft mindestens eine Methode der Schnittstelle auf

INTERFACES



INTERFACES

- Das Interface definiert eine Reihe von Methoden, die unabhängig von der Implementierung für die dauerhafte Speicherung von Kundendaten notwendig sind. Die Kundenverwaltung benutzt dieses Interface in einer ihrer Methoden, spielt also die Rolle der verwendenden Klasse



- Die Alternativen der Schnittstellen-Implementierung (`DateiKundenSpeicher` und `DBKundenSpeicher`) spielen die Rolle der implementierenden Klassen. Sie bieten je eine Realisierung zu jeder Methode, die in der Schnittstelle `IKundenSpeicher` vereinbart wurde

INTERFACES

- Verwendung der Schnittstelle an einer Beispiel-Methode, die Definition des Interfaces sowie eine Beispiel-Implementierung.
- (Die Anweisungen im Rumpf der implementierten Methoden sind aus Platzgründen abgeschnitten.)

Schnittstelle mit
vereinbarten
Methoden

```
public interface IKundenSpeicher {  
    public void neu(Kunde k);  
    public Kunde laden(int kundenNr);  
    public void aktualisieren(Kunde k);  
    public void löschen(int kundenNr);  
}
```

Implemen-
tierung der
Schnittstellen-
Methoden

```
public class DBKundenSpeicher implements IKundenSpeicher{  
    public void neu(Kunde k) { ... }  
    public Kunde laden(int kundenNr) { ... }  
    public void aktualisieren(Kunde k) { ... }  
    public void löschen(int kundenNr) { ... }  
}
```


INTERFACES

Verwenden von
DBKundenSpeicher
als konkrete
Implementierung
der Schnittstelle
IKundenSpeicher

Aufruf einer
Schnittstellen-
Methode

```
public class Kundenverwaltung {  
    private Kunde k1;  
    private Kunde k2;  
    ...  
    private IKundenSpeicher kundenSpeicher = new DBKundenSpeicher();  
  
    public void aktualisiereAlleKunden(){  
        kundenSpeicher.aktualisieren(k1);  
        kundenSpeicher.aktualisieren(k2);  
        ...  
    }  
}
```



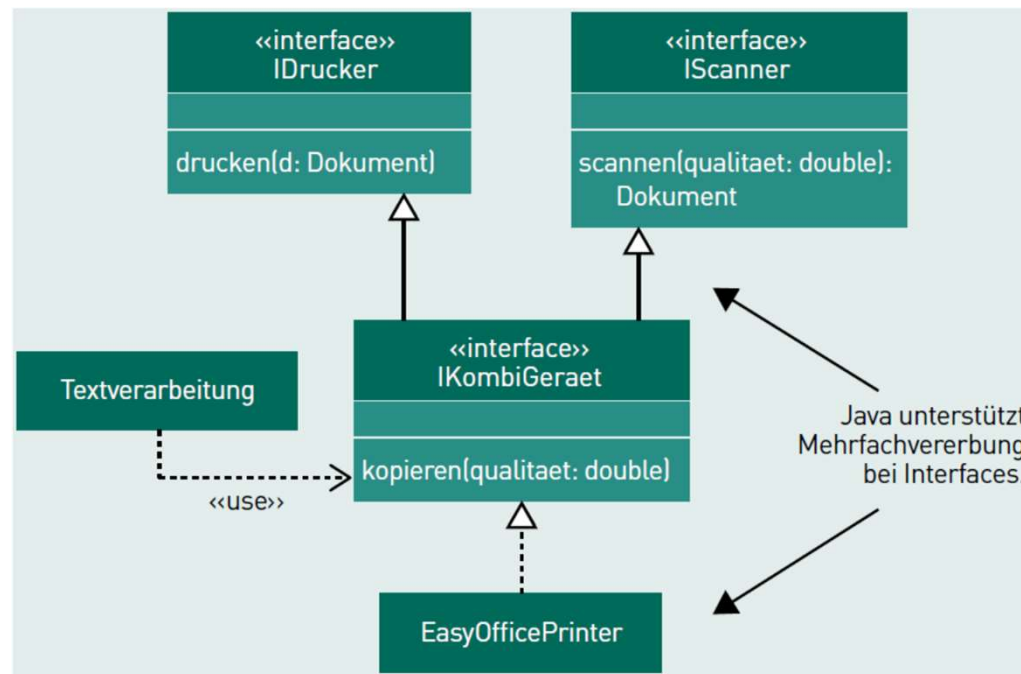
INTERFACES

- Weil die benötigten Methoden zur Speicherung von Kundendaten nun in einer Schnittstelle definiert sind, können sie ohne großen Aufwand beliebig ausgetauscht werden (Flexibilität)
- Umgekehrt kann die Implementierung zur Speicherung von Kundendaten prinzipiell an jeder weiteren Programmstelle eingesetzt werden, an der ein solches Interface benötigt wird (Wiederverwendbarkeit)

INTERFACES

- Interfaces können auch von anderen Interfaces erben. Auf diese Weise können Gemeinsamkeiten von Schnittstellen zusammengefasst und je nach Verwendungszweck ohne großen Wartungsaufwand erweitert werden.
- Bezogen auf Interfaces unterstützt Java außerdem das Konzept der „Mehrfachvererbung“.
- Das heißt, im Gegensatz zu Klassen können Interfaces sogar von mehr als einem Interface erben. Trotzdem wird mit `extends` das gleiche Schlüsselwort verwendet

INTERFACES



INTERFACES

```
public interface IKombiGeraet extends IDrucker, IScanner {  
    public void kopieren(double qualitaet);  
}  
  
public interface IDrucker {  
    public void drucken(Dokument d);  
}  
  
public interface IScanner {  
    public Dokument scannen(double qualitaet);  
}  
  
public class EasyOfficePrinter implements IKombiGeraet {  
    public void kopieren(double qualitaet) { ... }  
    public void drucken(Dokument d) { ... }  
    public Dokument scannen(double qualitaet) { ... }  
}
```



INTERFACES

- Parallelen -Interfaces und abstrakte Klassen:
 - Abstrakte Klassen können mit abstrakter Methoden Klassen-übergreifende Funktionalitäten festlegen
 - Bei Interfaces spielt die Vererbungshierarchie keine Rolle. Auch „unverwandte“ Klassen können einem gleichen Interface zugeordnet werden
 - Eine Klasse kann mehr als ein Interface implementieren
 - Bei Interfaces ist es verboten, implementierte und abstrakte Methoden zu vermischen. Ebenso wird durch das Verbot Attribute zu definieren verhindert, zustandsbehaftete Interfaces zu programmieren
 - Interfaces können default Methoden besitzen, welche einer Implementierten Methode nahe kommt.



INTERFACES

DEFAULT METHODEN



DEFAULT METHODEN

- Seit Java 8 kamen „Default“ Methoden in Interfaces hinzu.
 - Eine default Methode bringt ihre Implementierung gleich mit.
 - *Davon abgeleitete Klassen, müssen default Methoden somit nicht Konkretisieren.*

```
public interface IFressbar {  
  
    default boolean wirdgefressen(int bissen) {  
        for(int i = 0; i <= bissen; i++) {  
            System.out.println("Mampf Mampf");  
        }  
    }  
}
```


DEFAULT METHODEN

- Bei der Definition von Schnittstellen ist zu beachten, dass sie grundsätzlich zustandslos sein müssen. Es ist daher bis auf Konstanten nicht erlaubt, Attribute zu definieren
- Mithilfe des Schlüsselwortes implements kann eine Klasse anzeigen, welche Schnittstelle(n) sie realisiert. Damit verpflichtet sie sich, für jede Methode dieser Schnittstelle(n) eine Implementierung anzubieten

```
public interface IFressbar {  
  
    default void wirdgefressen(int bissen) {  
        for(int i = 0; i<=bissen; i++){  
            System.out.println("Mampf Mampf");  
        }  
    }  
}
```



DEFAULT METHODEN

- Damit geht aber ein Problem einher. Wenn zwei Interfaces, warum auch immer, identisch benannte Methoden zur Verfügung stellen und unsere Klasse diese implementieren, kommt es zur Überschneidung wo wir dann doch Default Methoden überschreiben müssen.

DEFAULT METHODEN

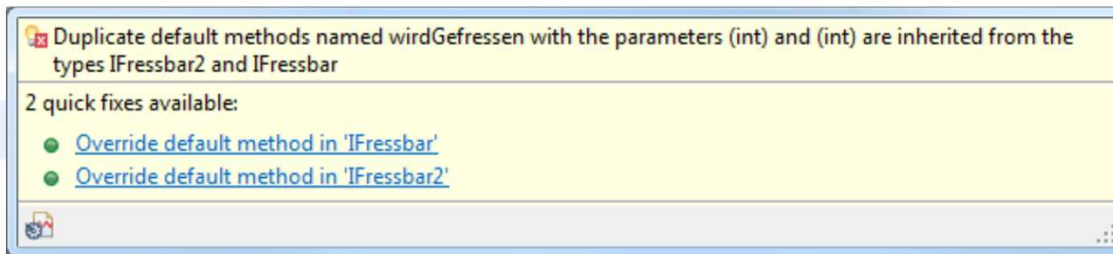
- Problem mit 2 Identisch benannte Defaultmethoden.

```
public interface IFressbar {  
  
    default void wirdgefressen(int bissen) {  
        for(int i = 0; i <= bissen; i++) {  
            System.out.println("Mampf Mampf");  
        }  
    }  
}
```

```
public interface IFressbar2 {  
  
    default void wirdGefressen(int bissen) {  
        for(int i = 0; i <= bissen; i++) {  
            System.out.println("Mampf2 Mampf2");  
        }  
    }  
}
```

```
public class VielFrucht implements IFressbar, IFressbar2{
```

```
}
```



DEFAULT METHODEN

- Lösung – Überschreiben der default Methode und Konkreter Aufruf der gewünschten default Methode

```
public interface IFressbar {  
  
    default void wirdgefressen(int bissen) {  
        for(int i = 0; i <= bissen; i++) {  
            System.out.println("Mampf Mampf");  
        }  
    }  
}
```

```
public interface IFressbar2 {  
  
    default void wirdGefressen(int bissen) {  
        for(int i = 0; i <= bissen; i++) {  
            System.out.println("Mampf2 Mampf2");  
        }  
    }  
}
```

```
public class VielFrucht implements IFressbar, IFressbar2 {  
  
    @Override  
    public void wirdGefressen(int bissen) {  
        // TODO Auto-generated method stub  
        IFressbar.super.wirdGefressen(bissen);  
        IFressbar2.super.wirdGefressen(bissen);  
    }  
}
```



INTERFACES

FUNCTIONAL INTERFACES & LAMBDA



FUNCTIONAL INTERFACES

- Interfaces mit nur einer abstrakten Methode werden auch „Functional Interfaces“ genannt.
- Die Annotation „@FunctionalInterface“ ist dabei optional anzugeben. Die Angabe zwingt den Compiler einen Fehler auszugeben sobald mehr als eine abstrakte Methode in das Interface geschrieben wird.
- Mit functional Interfaces ist die sogenannte Funktionale Programmierung möglich
 - - Dies geschieht in Java über die Lambda-Ausdrücke



FUNCTIONAL INTERFACES

- Lambda Ausdrücke ermöglichen es, funktionale Interfaces ohne anonyme Innere Klasse zu verwenden.