

Grundlagen der Programmierung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Vorlesungsskript zum Sommersemester 2020

11. Vorlesung (6. Juli 2020)



Dr. Jin Gerlach

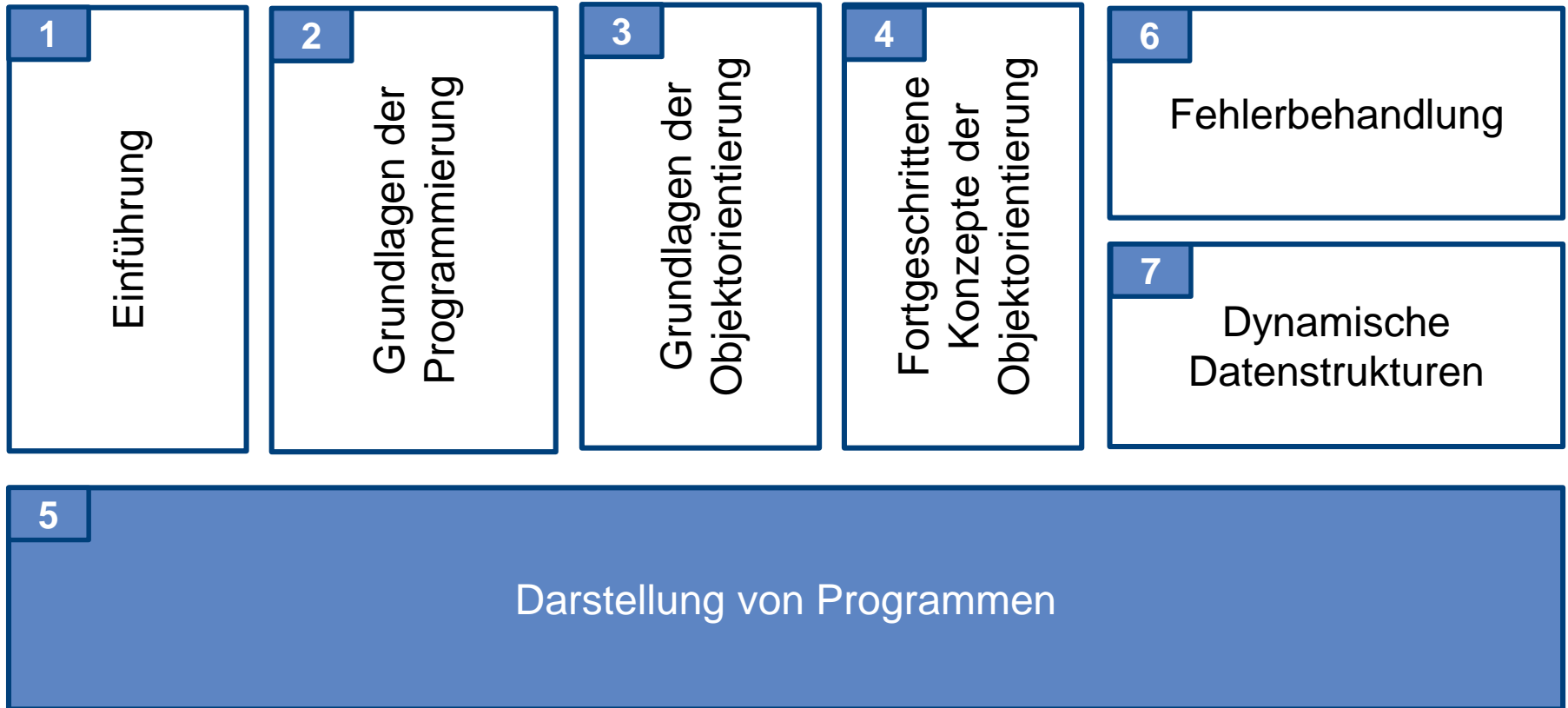
Christian Olt

Fachgebiet Wirtschaftsinformatik | Software & Digital Business

Fachbereich Rechts- und Wirtschaftswissenschaften

Technische Universität Darmstadt

Thematische Übersicht



Kapitel 5: Darstellung von Programmen

- Klassendiagramme in UML

Lernziele:

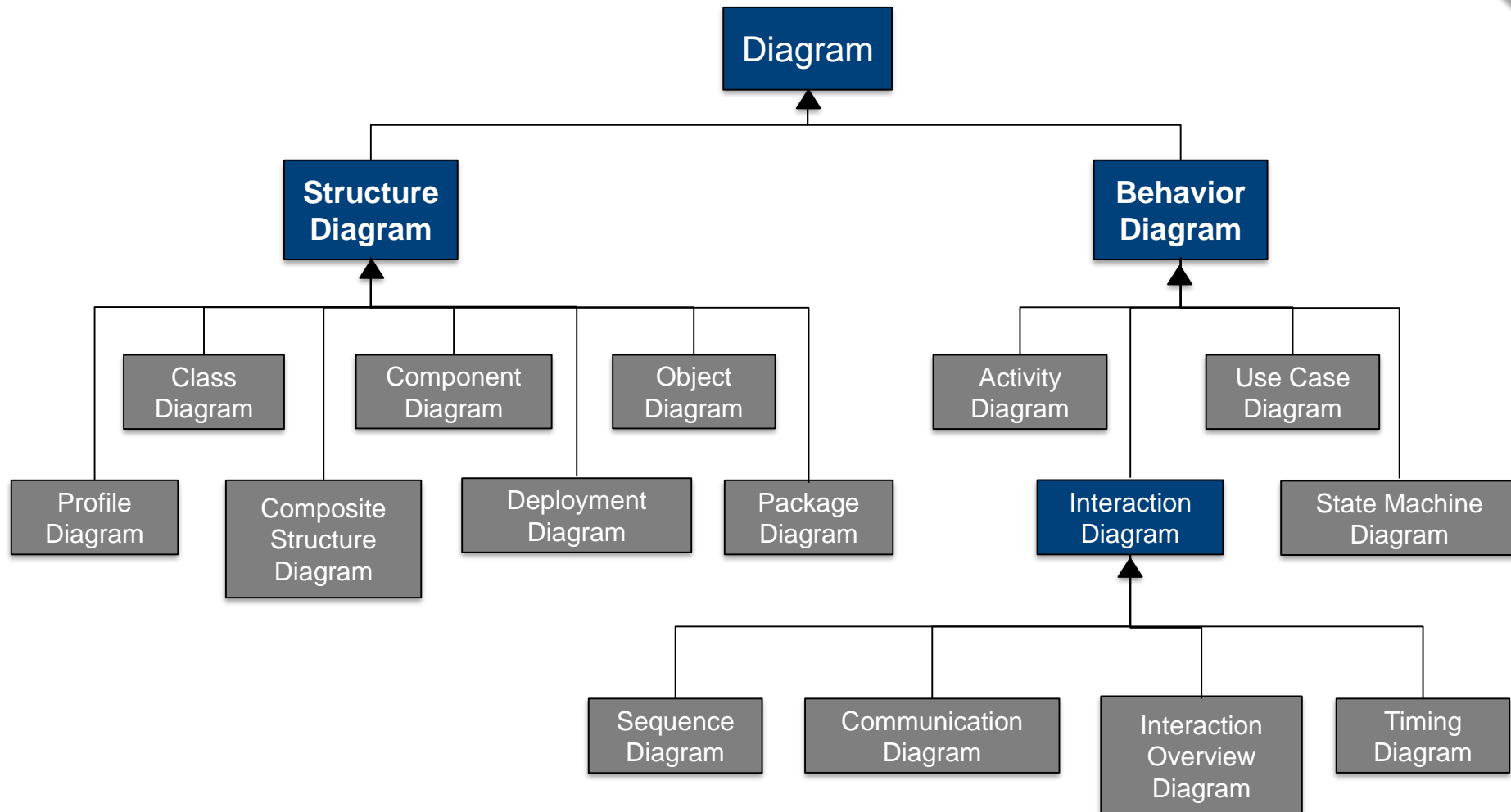
- Das Ziel eines UML-Klassendiagramms erklären können.
- Die Elemente eines UML-Klassendiagramms in Java kennen.
- Die drei Beziehungstypen Generalisierung, Realisierung und Assoziation in einem UML-Klassendiagramm kennen, unterscheiden und anwenden können.
- UML-Klassendiagramme zeichnen und interpretieren können.

Diagrammtypen der UML



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Wiederholung
Exkurs



- **Klassendiagramme** sind Strukturdiagramme und umfassen:
 - Klassen (sowohl konkrete als auch abstrakte Klassen) sowie Interfaces
 - deren Methoden und Attribute
 - Beziehungen zwischen den Klassen und Interfaces

Modellierung von Klassen

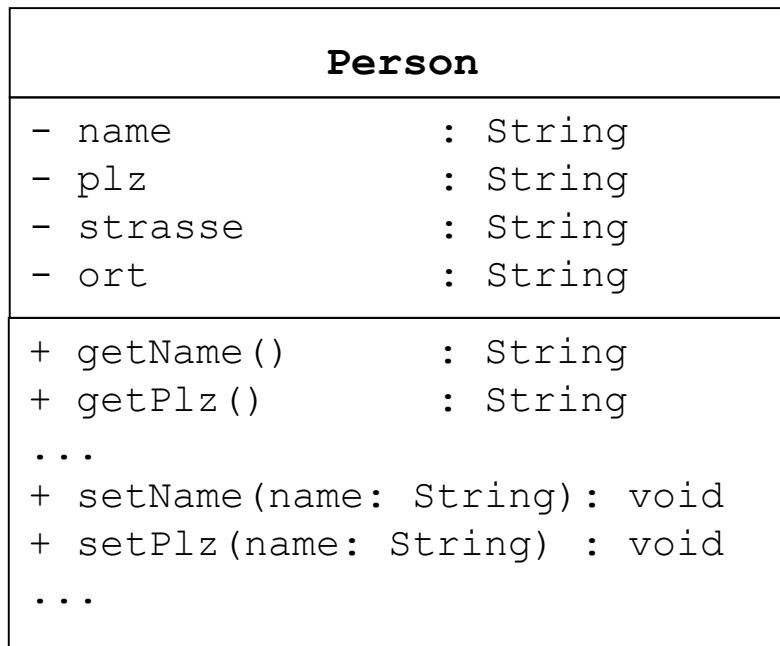
- Klassen werden als Rechtecke dargestellt
- Klassenname wird fettgedruckt (in Klausur nicht notwendig)
- Attribute und Methoden können angegeben werden
- Bereiche für Klassenname, Attribute und Methoden werden durch horizontale Linien getrennt
- Statische Elemente (Klassenattribute und -methoden) werden unterstrichen oder durch ein vorangestelltes `<<static>>` gekennzeichnet

EineKlasse	
- normalesAttribut	: Datentyp
- <u>statischesAttribut</u>	: <u>Datentyp</u>
+ <u>statischeMethode1 (...)</u>	: <u>Rückgabotyp</u>

Java-Sichtbarkeit	UML-Notation
private	-
package	~
protected	#
public	+

Eine Klasse	
...	
-	privateMethod() : int
~	packageMethod() : boolean
#	protectedMethod() : String
+	publicMethod() : byte

Beispiel: Klassendiagramm



Klassenname

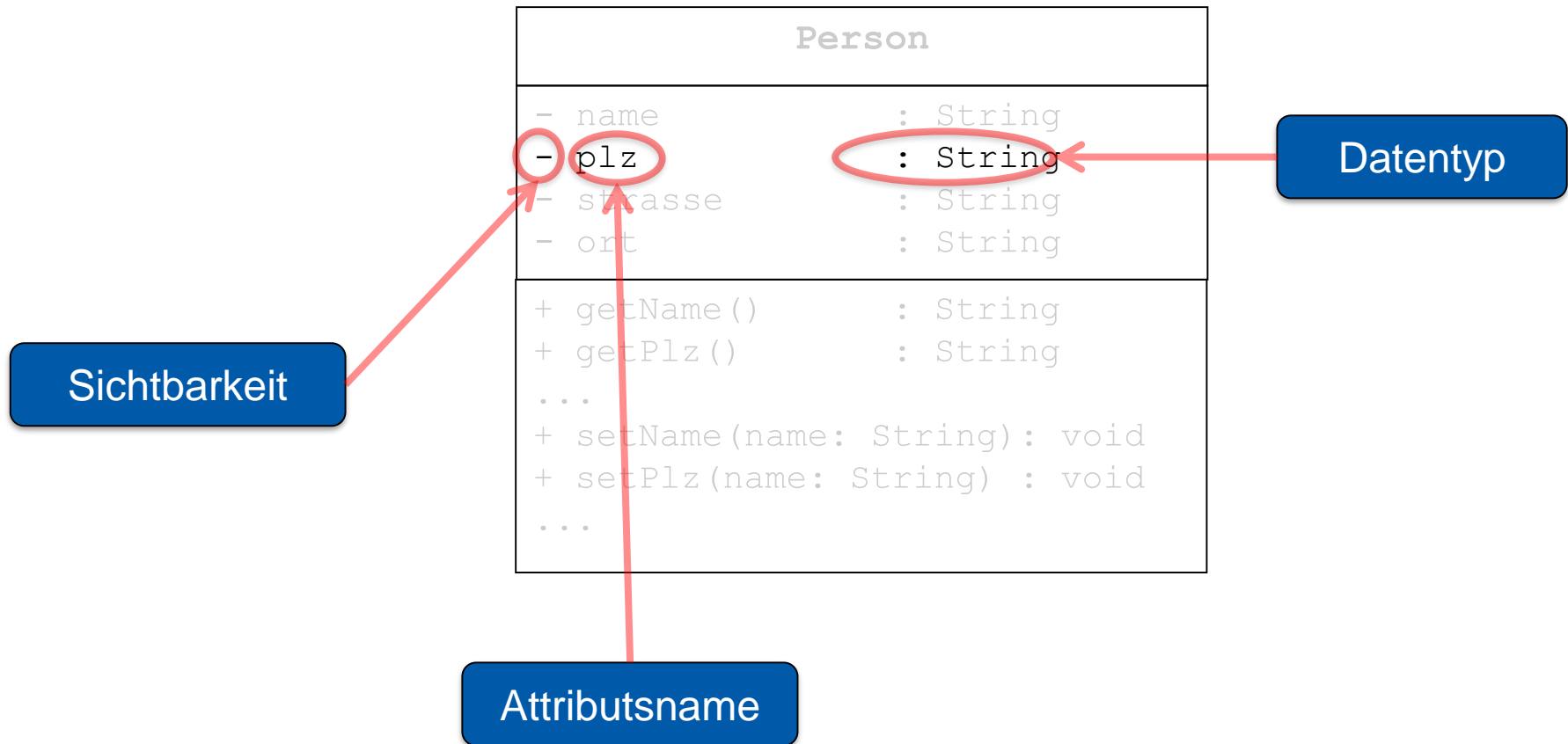
Attribute

Methoden

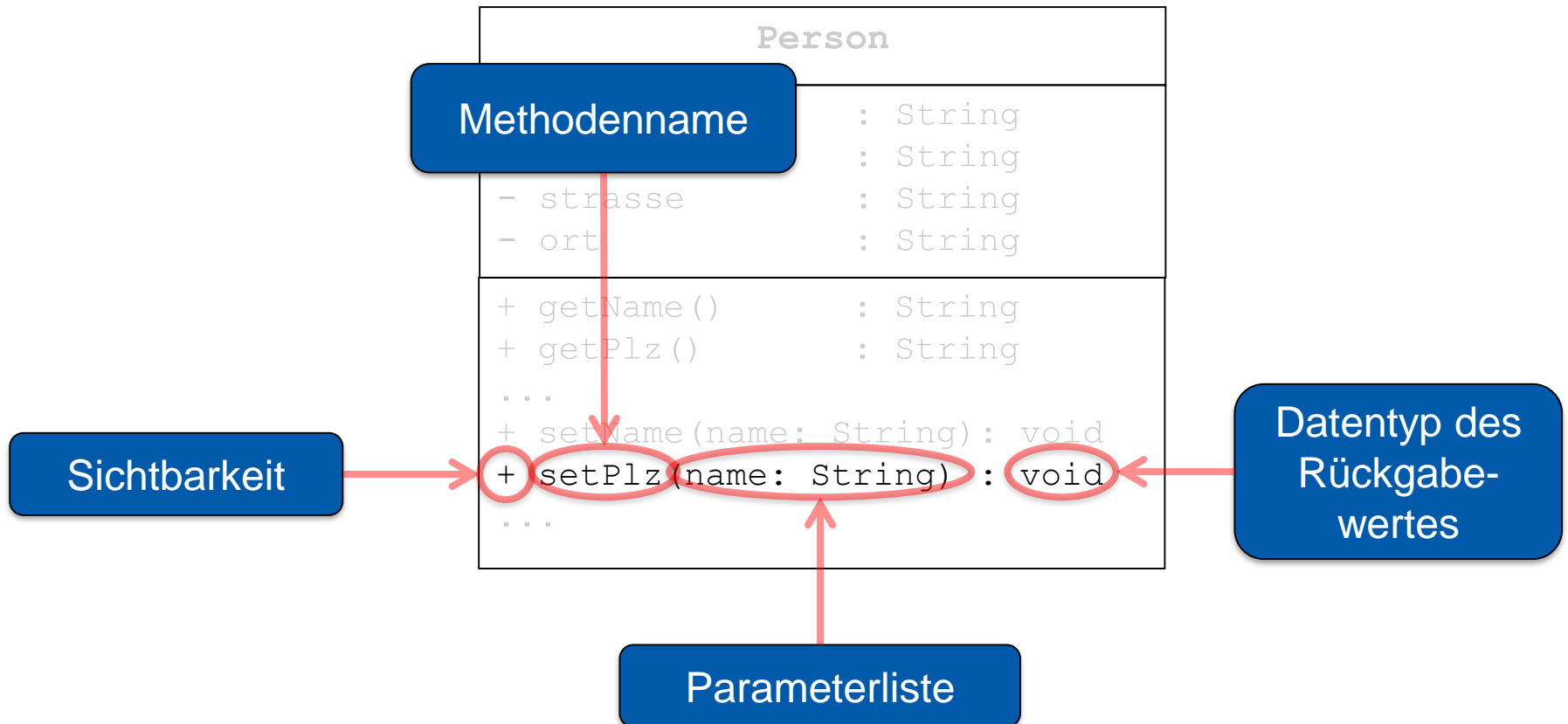
Beispiel: Attribute



TECHNISCHE
UNIVERSITÄT
DARMSTADT



Beispiel: Methoden



Abstrakte Klassen und Interfaces

- Abstrakte Klassen und Methoden werden kursiv geschrieben oder durch ein vorangestelltes `<<abstract>>` gekennzeichnet (zweite Variante für Klausur!)
- Interfaces werden durch ein vorangestelltes `<<interface>>` gekennzeichnet

<code><<interface>></code> EinInterface
...
+ <code>methode1(...)</code> : Rückgabotyp

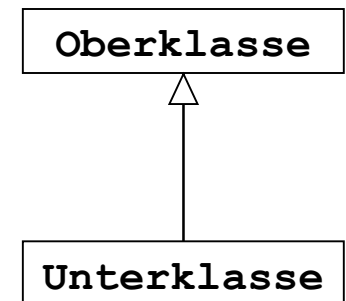
<code><<abstract>></code> EineAbstrakteKlasse
...
<code><<abstract>></code> + <code>methode1(...)</code> : Rückgabotyp

- Konstrukturen werden gekennzeichnet, indem vor die entsprechende Methode `«constructor»` geschrieben wird
- Der Rückgabetyp wird nicht angegeben

EineKlasse
+ «constructor» EineKlasse()

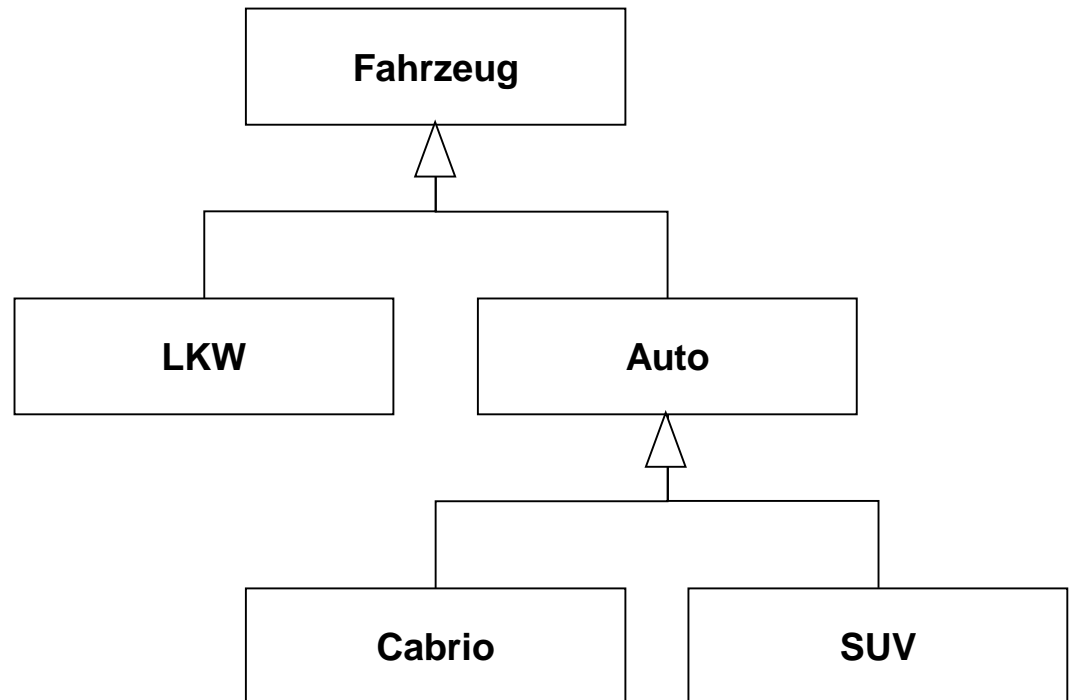
Beziehung I: Generalisierung

- Die *Generalisierung* ist eine Beziehung zwischen einem allgemeinen und einem speziellen Element.
- In Java entspricht dies der Vererbung (zwischen einer Ober- und einer Unterklasse) oder der Erweiterung (zwischen zwei Interfaces).
- Darstellung: Eine durchgezogene Verbindungslinie mit einem unausgefüllten Dreieck am Ende, das auf das allgemeine Element zeigt

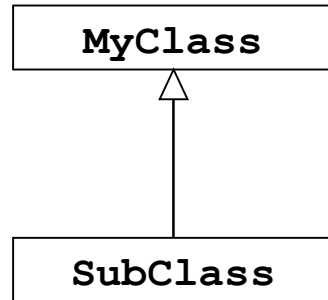


Transitive Vererbung

- Vererbung ist **transitiv**:
Von **Sub-Klassen** können wiederum neue **Sub-Sub-Klassen** abgeleitet werden
- Diese **Sub-Sub-Klassen** erben auch die Attribute und Methoden, welche die eigene **Ober-Klasse** von ihrer **Ober-Klasse** geerbt hat

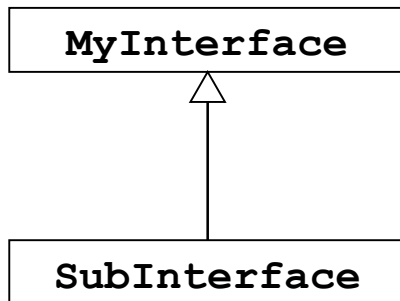


Beispiel: Generalisierung



```
public class MyClass {  
    ...  
}
```

```
public class SubClass  
    extends MyClass {  
    ...  
}
```

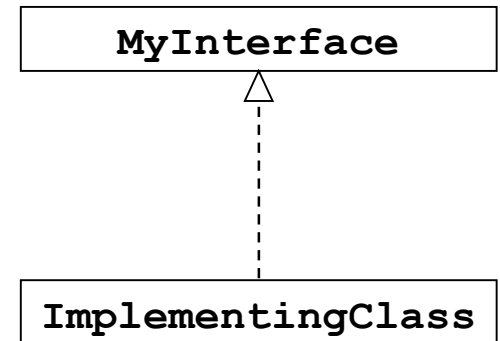


```
public interface MyInterface {  
    ...  
}
```

```
public interface SubInterface  
    extends MyInterface {  
    ...  
}
```

Beziehung II: Realisierung

- Die **Realisierung** ist eine Beziehung zwischen einem Element, das Anforderungen stellt und einem anderen Element, das die Anforderungen erfüllt.
- In Java entspricht dies der **Implementierung eines Interfaces durch eine Klasse**.
- Darstellung durch eine gestrichelte Linie zwischen der Klasse und dem Interface, mit einem unausgefüllten Dreieck am Ende, das auf das Interface zeigt.

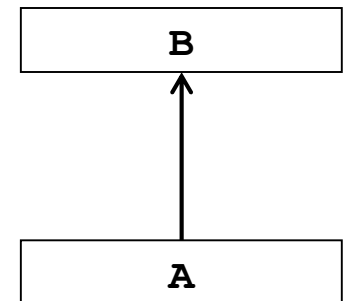


```
public interface MyInterface
{
    ...
}

public class ImplementingClass
    implements MyInterface {
    ...
}
```


Beziehung III: Assoziation

- Die **Assoziation** ist eine Zugriffsbeziehung zwischen zwei Elementen.
- In Java entspricht dies dem **direkten Zugriff** auf eine andere Klasse (z. B. Verwendung einer anderen Klasse als Datentyp).
- Darstellung als eine durchgezogene Linie mit einer offenen Pfeilspitze am Ende, die vom zugreifenden auf das zugegriffene Element zeigt.



Beispiele: Assoziation

```
public class UsingClass {  
    public static void main(String[] args) {  
        System.out.println(Math.exp(2));  
    }  
}
```

UsingClass



Math

```
public class Class1 {  
    private Class2 attribute;  
    ...  
}
```

```
public class Class2 {  
    private Class1 attribute;  
    ...  
}
```

Class1

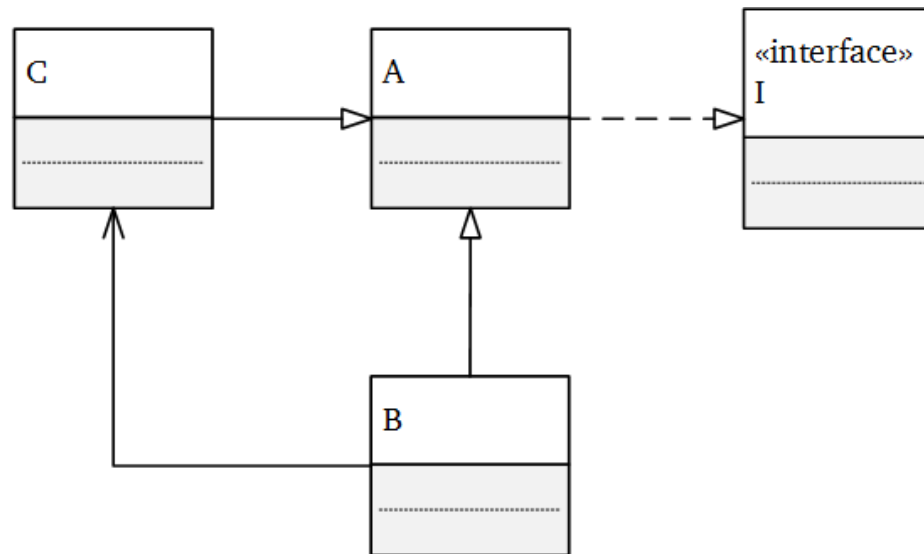


Class2

Aufgabe: Klassenhierarchie



Gegeben ist die folgende Klassenhierarchie eines Java-Programms.
Welche Werte können in der folgenden Anweisung anstelle von
\$DATENTYP\$ eingesetzt werden?
\$DATENTYP\$ myVar = new C();



Thematische Übersicht



TECHNISCHE
UNIVERSITÄT
DARMSTADT

1

Einführung

2

Grundlagen der
Programmierung

3

Grundlagen der
Objektorientierung

4

Fortgeschrittene
Konzepte der
Objektorientierung

6

Fehlerbehandlung

7

Dynamische
Datenstrukturen

5

Darstellung von Programmen

Kapitel 6: Fehlerbehandlung

- Das „Exceptions“-Konzept
- „Werfen“ und „fangen“ und weiterleiten von Exceptions in Java
- Exceptions erstellen und verwenden

Lernziele:

- Exceptions werfen, fangen und weiterleiten können.
- Eigene Exceptions erstellen und verwenden können.



Folgen von Programmierfehlern

- 1996: Jungfernflug der Ariane 5
 - Kursabweichung nach 37 Sekunden
 - Einleitung der Selbstzerstörung
 - 370 Mio. Schaden
- 2010: Dow-Jones-Index verliert in 10 Minuten 6% seines Wertes aufgrund eines Systemfehlers
- 2010: Fluggesellschaft verschenkt versehentlich Tausende von Online-Tickets aufgrund eines Programmierfehlers
- 2016: Marssonde „Schiaparelli“ stürzt ab, da vermutlich die Software für Navigation und Höhenmesser fehlerhaft war
- 2018: Programmierfehler kostet Hamilton den Sieg in Melbourne

Exception Handling in Java

- **Exception Handling** ist das objektorientierte Konzept zur Behandlung von „Ausnahmesituationen“ (d.h. Fehlern)
- Ausnahmebehandlung ist Bestandteil der Programmiersprache Java:
 - Hinweis: Exceptions sind „gewöhnliche“ Objekte
 - Exceptions werden erstellt (**geworfen**), wenn eine Fehlersituation auftritt
 - Die Bearbeitung des Programmcodes wird an dieser Stelle abgebrochen und es besteht die Möglichkeit, die Exception **abzufangen** und zu behandeln

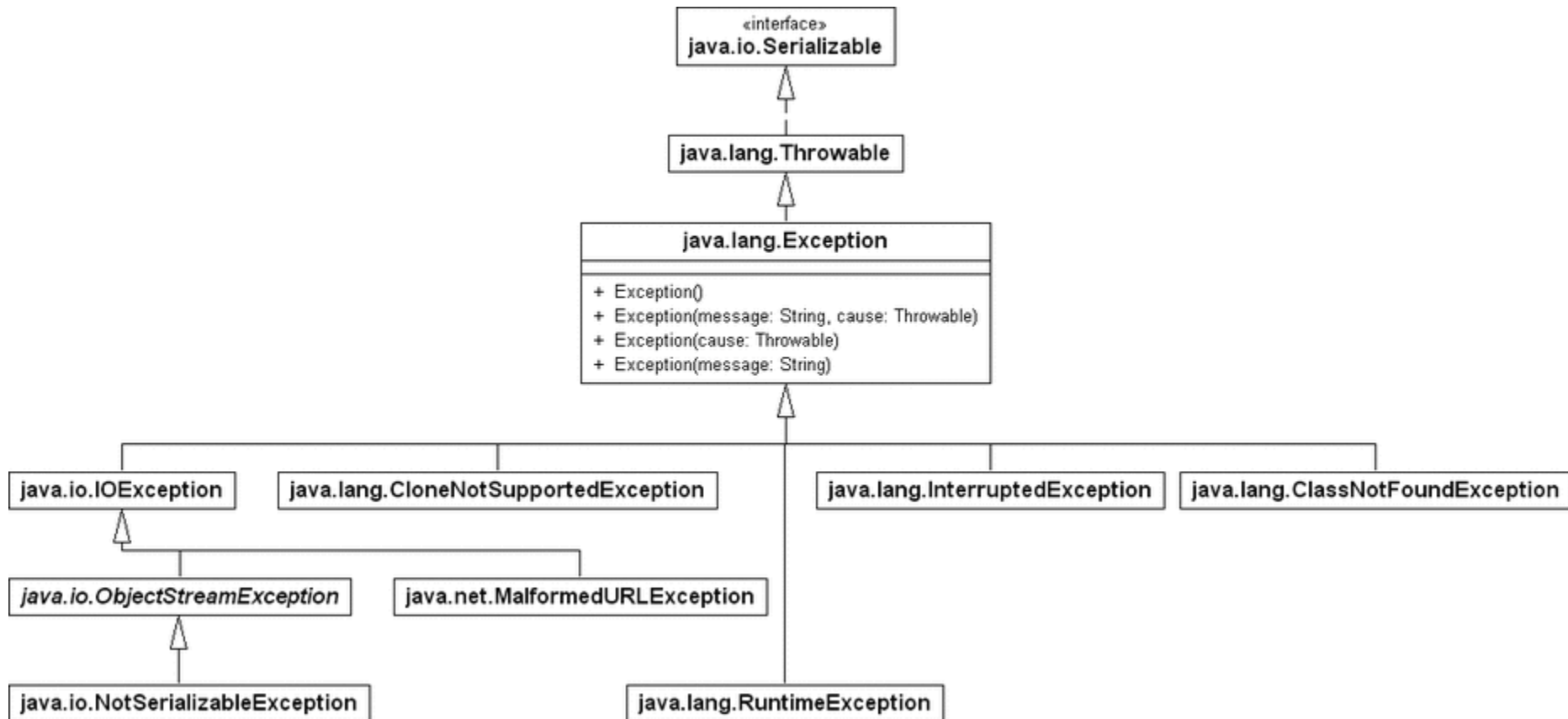


Exception-Hierarchie in Java

- Java verfügt über eine Klassenhierarchie von Exceptions
- Alle Exceptions erben von der Klasse `java.lang.Exception`
- Verschiedene Exception-Klassen bedeuten unterschiedliche Fehlerarten
 - `ArithmeticException`, z. B. für Division durch 0
 - `ArrayIndexOutOfBoundsException`, beim Zugriff auf ein Feldelement mit falschem Index
 - `IOException`, allgemein Fehler bei der Ein-/Ausgabe
 - `FileNotFoundException`, beim Zugriff auf eine nicht vorhandene Datei



Klassenhierarchie für Exceptions



Beispiel:

ArrayIndexOutOfBoundsException



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
public static void main(String[] args) {  
    int[] zahlen = {5, 10, 15};  
    System.out.println(zahlen[3]);  
}
```

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
at winf2.extest.Input.main(Input.java:12)

Exceptions in Java (1/3)

```
try {  
    // Geschuetzter Block  
} catch (ArithmeticException ex1ObjName) {  
    // Behandlung einer ArithmeticException  
} catch (IllegalArgumentException ex2ObjName) {  
    // Behandlung einer IllegalArgumentException  
} finally {  
    // Abschliessend auszufuehrender Code  
}
```

try-Block

catch-Blöcke

finally-Block

- Der `try`-Block enthält den Code, bei dem Exceptions auftreten können
- Tritt eine Exception auf, wird der `try`-Block an der Stelle abgebrochen (die nachfolgenden Befehle im `try`-Block werden nicht mehr ausgeführt)
- Es wird zu einem `catch`-Block gesprungen

Exceptions in Java (2/3)



TECHNISCHE
UNIVERSITÄT
DARMSTADT

```
try {  
    // Geschuetzter Block  
} catch (ArithmeticException ex1ObjName) {  
    // Behandlung einer ArithmeticException  
} catch (IllegalArgumentException ex2ObjName) {  
    // Behandlung einer IllegalArgumentException  
} finally {  
    // Abschliessend auszufuehrender Code  
}
```

Exception-
Datentyp und
Objektname

- Es wird der erste `catch`-Block aufgerufen, der dem Datentyp der Exception entspricht
- Im `catch`-Block stehen Anweisungen zur Fehlerbehandlung
- Es können beliebig viele `catch`-Blöcke angegeben werden

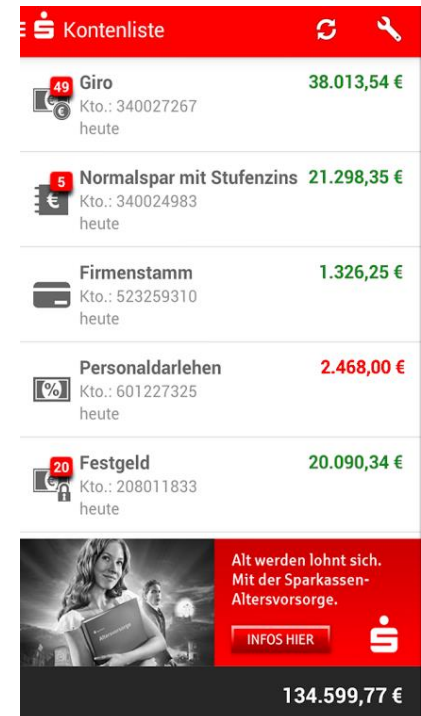
Exceptions in Java (3/3)







```
try {  
    // Geschuetzter Block  
} catch (ArithmeticException ex1ObjName) {  
    // Behandlung einer ArithmeticException  
} catch (IllegalArgumentException ex2ObjName) {  
    // Behandlung einer IllegalArgumentException  
} finally {  
    // Abschliessend auszufuehrender Code  
}
```

- Der `finally`-Block ist optional und wird **abschließend** ausgeführt, unabhängig davon, ob eine Exception **aufgetreten ist oder nicht**

Beispiel: Exception Handling in Java

```
public static void main(String[] args) {  
  
    int[] kontostaendeInTsd = {38, 21, 1, -2, 20};  
    int index = 5; // Hier können die Werte geändert werden !!!  
  
    try {  
        System.out.println(kontostaendeInTsd[index]);  
    }  
    catch (ArrayIndexOutOfBoundsException ex) {  
        System.out.println("Falscher Index!");  
    }  
}
```



Kontenliste	
 Giro Kto.: 340027267 heute	38.013,54 €
 Normalspar mit Stufenzins Kto.: 340024983 heute	21.298,35 €
 Firmenstamm Kto.: 523259310 heute	1.326,25 €
 Personaldarlehen Kto.: 601227325 heute	2.468,00 €
 Festgeld Kto.: 208011833 heute	20.090,34 €
 <div>Alt werden lohnt sich. Mit der Sparkassen- Altersvorsorge. INFOS HIER</div>	
134.599,77 €	

Weiterleiten von Exceptions

- Soll eine Exception nicht an Ort und Stelle mit try und catch behandelt werden, kann sie auch an den Aufrufer weitergeleitet werden
- Dies geschieht im Methoden-Kopf, indem das Schlüsselwort `throws` sowie der Exception-Datentyp angegeben wird
- Beispiel:

```
public void setBalance(double newBalance) throws Exception { ...
```
- Der Aufrufer muss dann die geworfene/weitergeleitete Exception behandeln oder diese ebenfalls weiterleiten

Werfen von Exceptions

- Man kann selbst festlegen, ob durch eine Anweisung eine bestimmte Exception geworfen werden soll
- Hierzu wird mit dem Schlüsselwort `throw` ein Exception-Objekt geworfen
- Syntax: `throw new` Exception-Typ (z. B. Exception)
- Durch die Erzeugung des Exception-Objekts wird der Programmfluss abgebrochen und es muss eine Fehlerbehandlung erfolgen

Beispiel: Werfen von Exceptions

```
public void setBalance(double newBalance) throws Exception {  
    if (currentBalance < 0) {  
        throw new Exception(  
            "Current balance has an invalid state: " + currentBalance);  
    }  
  
    if (newBalance < 0) {  
        throw new IllegalArgumentException(  
            "Balance cannot be negative!");  
    }  
  
    // ...  
}
```

Weiterleiten von Exceptions

Schlüsselwort
throw zum
Werfen einer
Exception

Eigene Exceptions erstellen

```
public class PLZException extends Exception {  
  
    public PLZException(String msg) {  
        super(msg);  
    }  
  
}
```

Konstruktor von `Exception` belegt ein Attribut für eine Fehlernachricht. Dieses wird hier weitergereicht.

Eigene Exceptions sind einfache Klassen, die von der `Exception`-Klasse erben.

Beispiel: Eigene Exceptions verwenden



```
public class Adresse {  
    private String plz;  
  
    public String getPlz() {  
        return plz;  
    }  
  
    public void setPlz(String plz) throws PLZException {  
        if (plz.length() == 5) {  
            this.plz = plz;  
        } else {  
            throw new PLZException("PLZ muss genau fuenf Stellen haben!");  
        }  
    }  
}
```

Checked vs. Unchecked Exceptions

- Exceptions, die von der Klasse `RuntimeException` abgeleitet sind, werden auch „**Unchecked Exceptions**“ genannt und müssen nicht (können aber) explizit behandelt werden, z. B.
`ArrayIndexOutOfBoundsException`
- Alle anderen Exceptions (**Checked Exceptions**) müssen vor dem Kompilieren explizit behandelt werden
 - try-catch-Behandlung
 - oder Weiterleiten der Exception an Aufrufer durch **throws**