Barry Linnert
Institut für Informatik

Freie Universität Berlin

Programmieren

Barry Linnert

Sommersemester 2020

Gliederung der heutigen Vorlesung

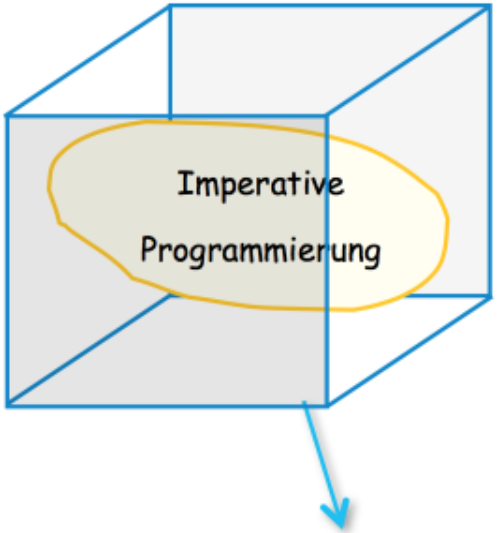
- Kurze Wiederholung
- Schnittstellen (Interfaces)
  - Beispiel: Schnittstelle als Referenztyp
  - Vererbung von Schnittstellen
  - Abstrakte Klasse vs. Schnittstelle
- Geschachtelte Klassen (Inner Classes)
  - Elementklassen
  - Lokale Klassen
  - Statisch geschachtelte Klassen
- Zusammenfassung

Freie Universität Berlin

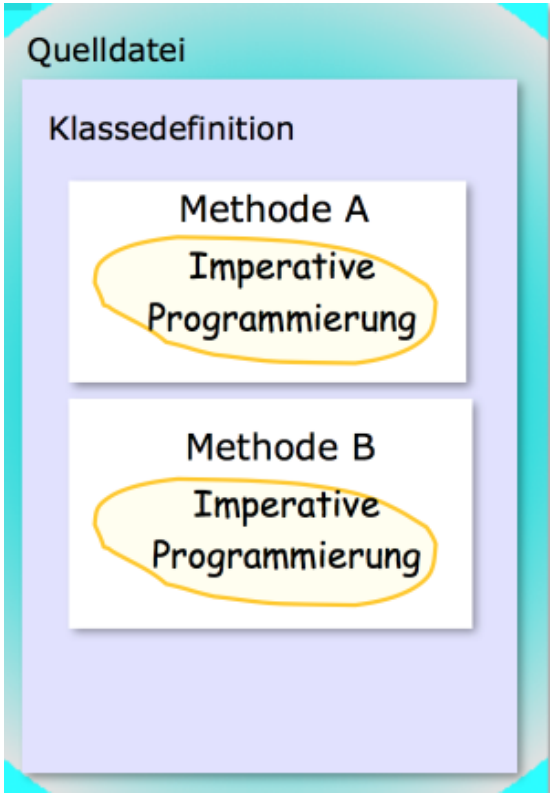
Schnittstellen und Geschachtelte Klassen

WIEDERHOLUNG

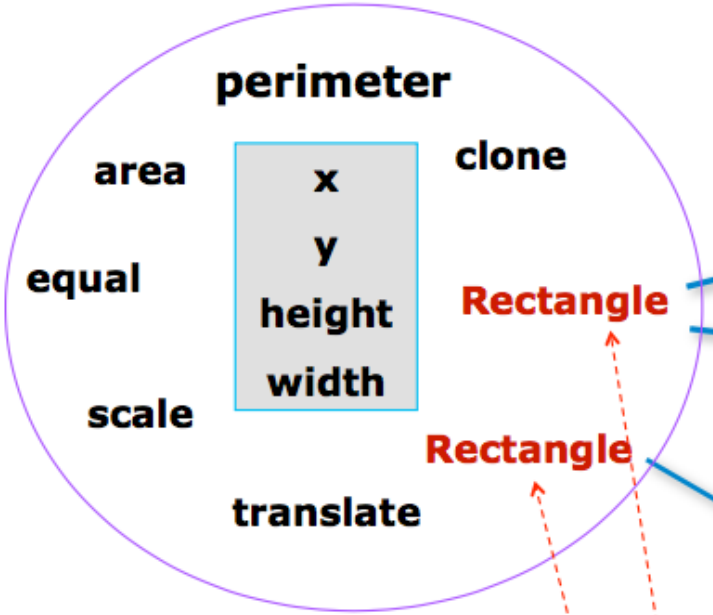
Imperative Grundbestandteile von Java



Der imperative Bestandteil eines Java-Programms befindet sich innerhalb der Methoden.



Rectangle-Klasse



Rectangle-Objekte

rectangle_A

x = 0
y = 0
height = 10
width = 10

area
scale
translate
...

rectangle_B

x = 0
y = 0
height = 10
width = 10

area
scale
translate
...

rectangle_C

x = 33
y = 45
height = 15
width = 15

area
scale
translate
...

Person p1 = new Person();

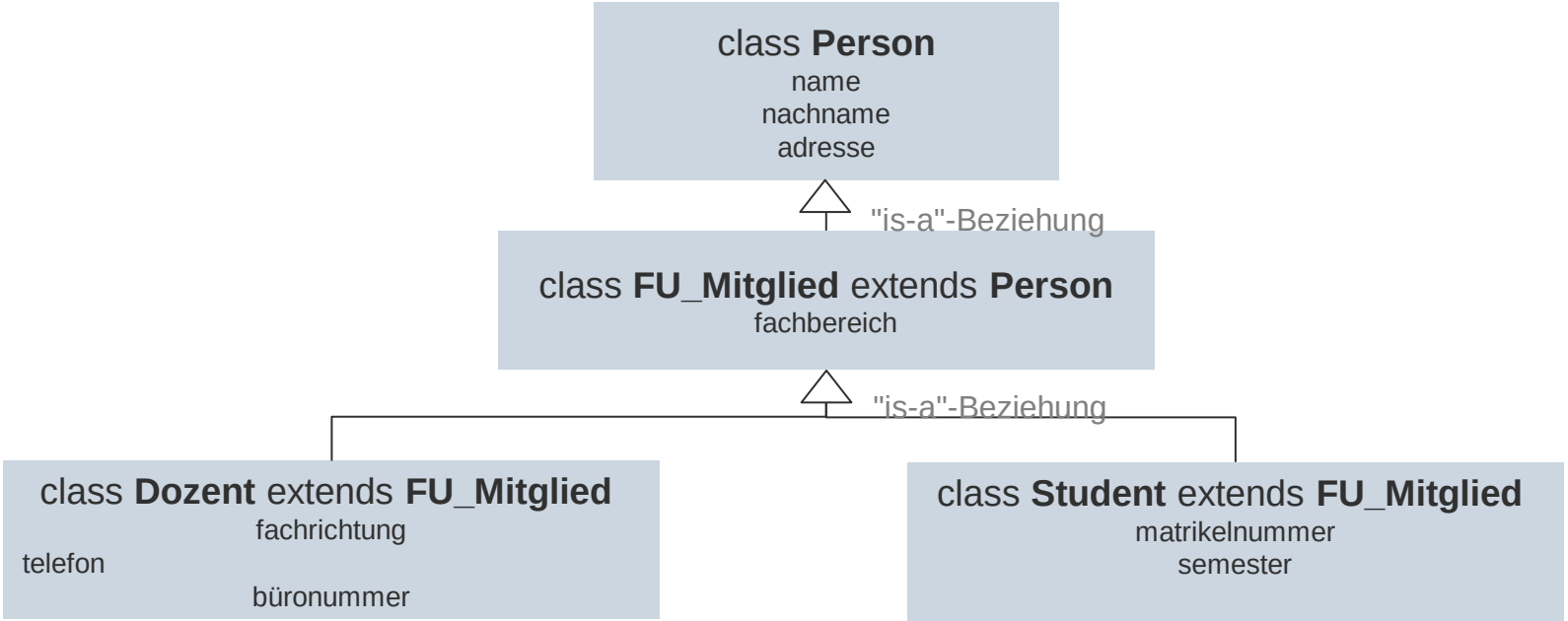
In **Schritt 1** wird die Referenzvariable p1 vom Typ Person angelegt und mit null initialisiert.

In Schritt 2 wird durch new Person der new-Operator aufgerufen und die Klasse Person instanziiert, d.h. es wird ein Objekt der Klasse Person auf dem Heap erzeugt.

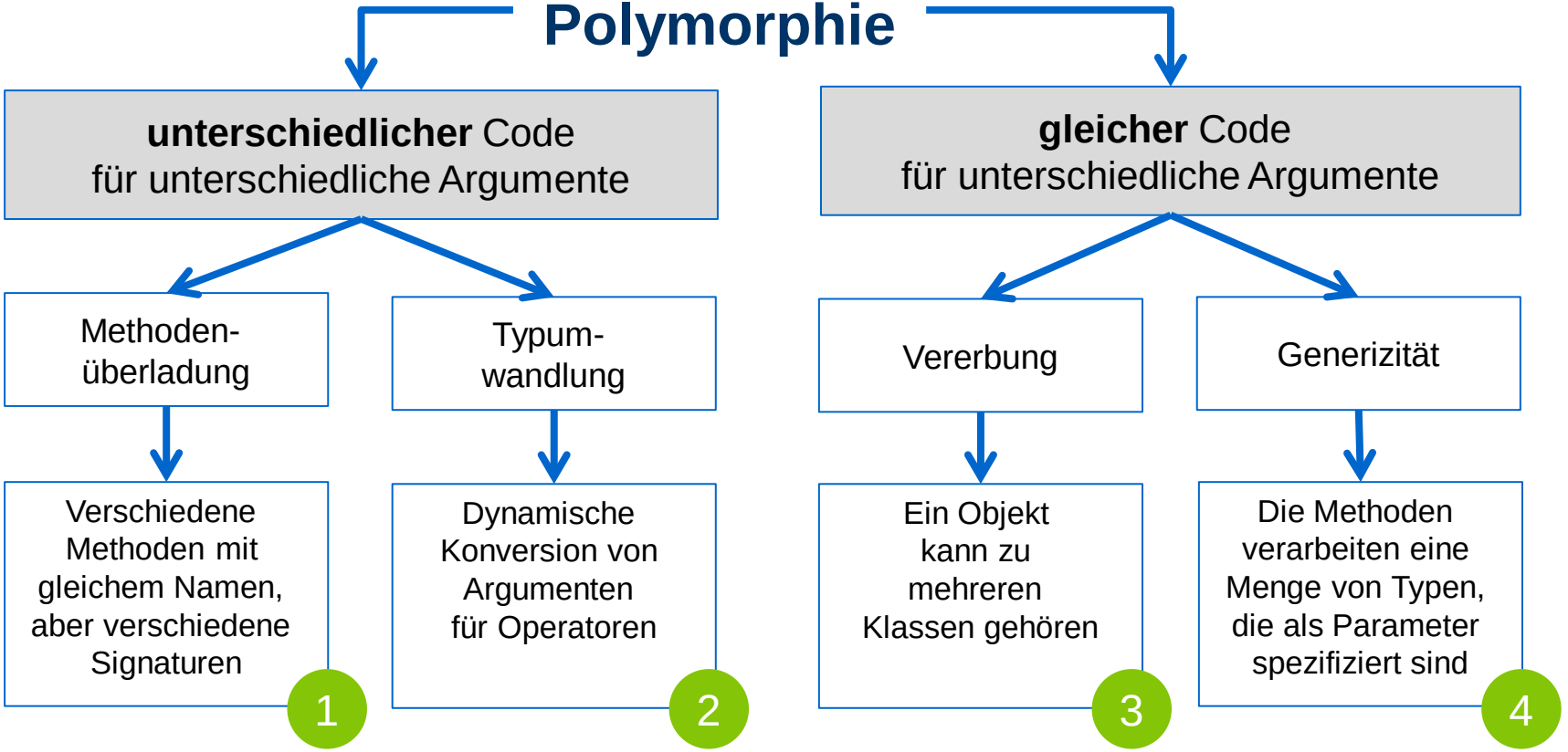
In Schritt 3 erfolgt die **Initialisierung des Objektes**. Es werden Default-Initialisierungen der Instanzvariablen des Objekts durchgeführt. Anschließend wird der Konstruktor aufgerufen.

In **Schritt 4** gibt der new-Operator eine **Referenz** auf das neu im Heap erzeugte Objekt zurück, welche **der Referenzvariablen** p1 **zugewiesen** wird.

Vererbung


class **Person**
name
nachname
adresse
"is-a"-Beziehung
class **FU_Mitglied** extends **Person**
fachbereich
"is-a"-Beziehung
class **Dozent** extends **FU_Mitglied**
fachrichtung
telefon
büronummer
class **Student** extends **FU_Mitglied**
matrikelnummer
semester

Polymorphie



Freie Universität Berlin

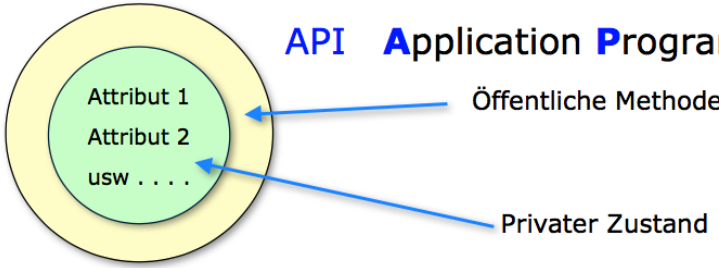
Schnittstellen und Geschachtelte Klassen

SCHNITTSTELLEN (INTERFACES)

Trennung von Spezifikation und Implementierung

- Eine gute Programmiersprache sollte das **Programmieren im Großen** – sprich den Entwurf – unterstützen.
- Die **Implementierung** einer Schnittstelle stellt eine **Verfeinerung des Entwurfs** dar.
- Die Unterstützung des Entwurfs erfolgt in Java mit Hilfe von Schnittstellen (Interface).
- Merke: Eine Klasse beinhaltet– sofern sie nicht abstrakt ist – den Entwurf und die Implementierung, d.h. auch die Methodenrümpfe.

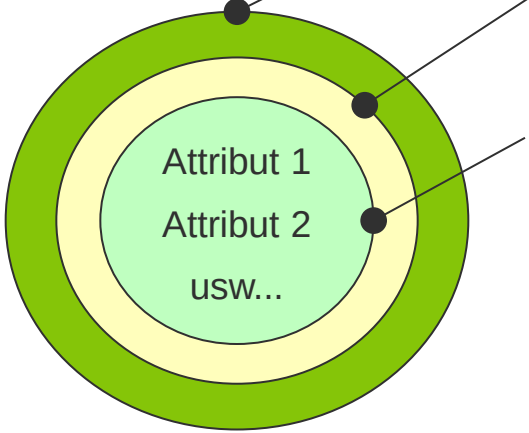
Methodenköpfe als Schnittstellen



API **A**pplication **P**rogramming **I**nterface.

Öffentliche Methoden (API)

Privater Zustand



Methodenkopf

Methodenrumpf

Daten

Motivation zum Einsatz von Interfaces in Java

- Um die Probleme der Namenskollisionen zu vermeiden wurde in Java mehrfache Vererbung abgeschafft.

- Im Java wird eine beschränkte mehrfache Vererbung mit Hilfe von Interfaces simuliert.

- **Merke:** Eine Klasse im Java kann von **einer** Oberklasse vererben, aber gleichzeitig **mehrere** Interfaces implementieren.

Interfaces in Java

- In Java sind Interfaces (Schnittstelle) sowohl ein **Abstraktionsmittel** (zum Verbergen von Details einer Implementierung) als auch ein **Strukturierungsmittel** zur Organisation von Klassenhierarchien!

- Eine Interface in Java legt eine **minimale Funktionalität** (Methoden) fest, die in einer implementierenden Klasse vorhanden sein soll.

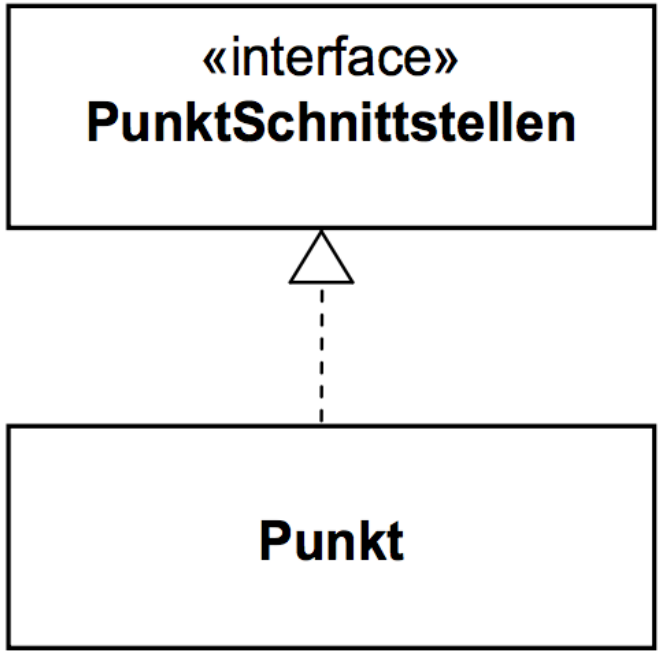
- Eigenschaften
  - Interfaces sind vollständig abstrakte Klassen.
  - In Interfaces sind keine Instanzvariablen deklariert.
  - Alle Methoden sind implizit **abstract** und **public**, aber keine ist implementiert.
  - Eine Klasse kann mehrere Schnittstellen implementieren.

Beispiel

```
interface PunktSchnittstellen {
   public int getX(); // Methode, um den x-Wert abzuholen
   public void setX (int i);// Methode, um den x-Wert zu setzen
}

public class Punkt implements PunktSchnittstellen {
   private int x; //x-Koordinate vom Typ int
   public int getX() {
      return x;}
   public void setX (int i) {
      x = i;}
}
```

Alle Methoden der Schnittstellen müssen in der Klasse implementiert sein, wenn die Klasse instanziierbar sein soll.

Beispiel: Darstellung in UML



Aufbau einer Schnittstelle

- Eine **Schnittstellendefinition** aus zwei Teilen:
  - der **Schnittstellendeklaration** und
  - dem Schnittstellenkörper mit Konstantendefinitionen und Methodendeklarationen.

```
public interface Name {

   public void methode1();
   public void methode2();
}
```

Schnittstellendeklaration

Schnittstellenkörper

Elemente einer Schnittstellendeklaration

- Optionaler Zugriffsmodifikator public (wird public nicht angegeben, so wird der Zugriffsschutz default verwendet, was bedeutet, dass die Sichtbarkeit auf das eigene Paket beschränkt ist)

- Schlüsselwort **interface** und dem Schnittstellennamen

- Optional: Schlüsselwort **extends** und durch Kommata getrennte Schnittstellen, von denen abgeleitet wird

`public`	optional
`interface Schnittstellenname`	zwingend erforderlich
`extends S1, S2, . . . , Sn`	optional ableiten von anderen Schnittstellen `S1` bis `Sn`

Elemente des Schnittstellenkörpers

- Der Schnittstellenkörper enthält:
  - Konstantendefinitionen und
  - Methodendeklarationen.

- Optional: Die explizite Angabe von public und abstract bei der Methodendeklaration

- Konstanten in Schnittstellen können als Übergabeparameter für eine Schnittstellenmethode verwendet werden. Jede Konstantendefinition in einer Schnittstelle muss einen Initialisierungsausdruck besitzen. Zum Beispiel: public static final int KONSTANTENNAME = 0;

Verwenden von Schnittstellen

- Implementiert eine Klasse eine Schnittstelle, so muss sie alle Methoden der Schnittstelle implementieren, wenn sie instanziiert werden soll – d. h. wenn von ihr Objekte geschaffen werden sollen.

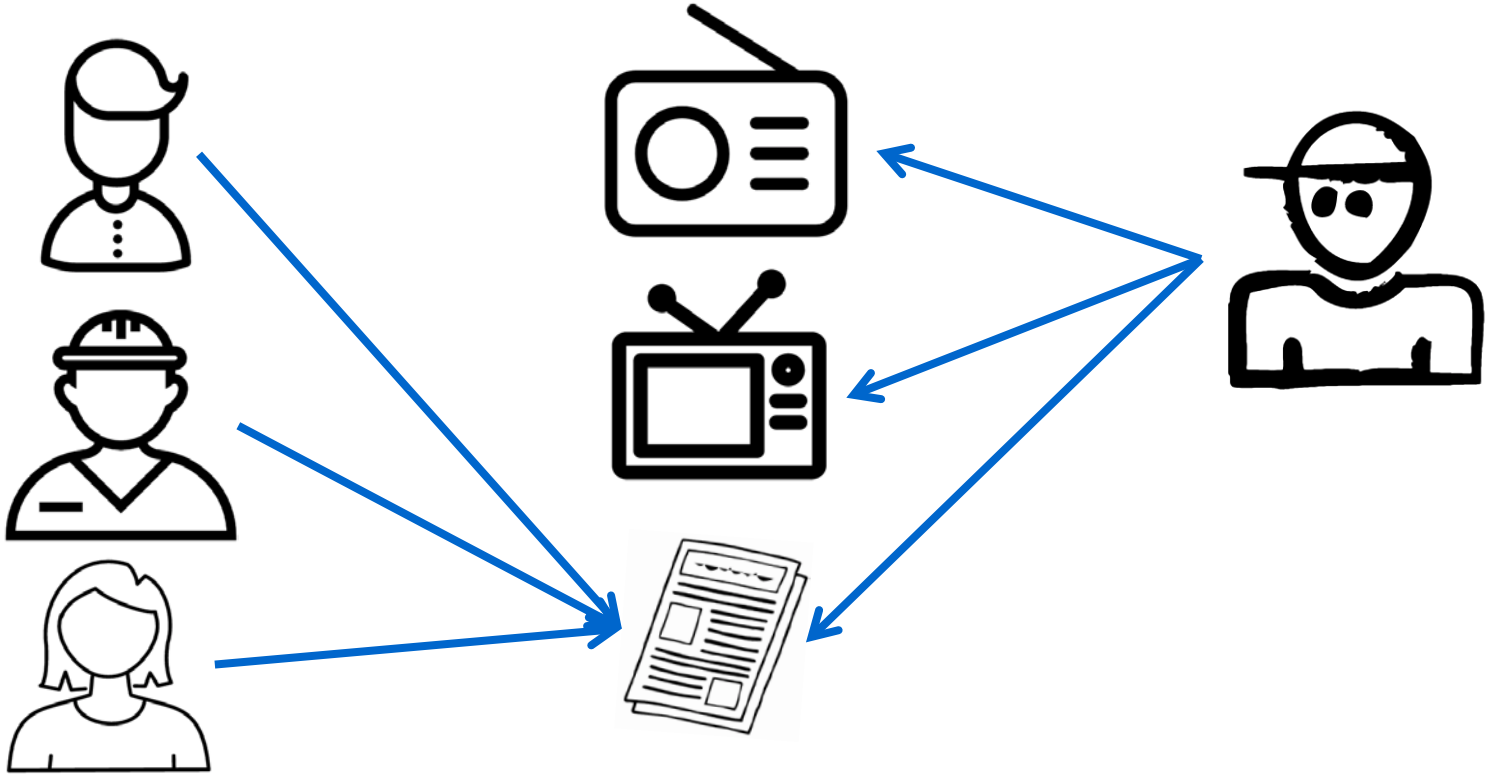
- Falls eine Klasse nicht alle Methoden implementiert, muss die Klasse als abstrakt deklariert werden und ist damit nicht instanziierbar.

- Eine Schnittstelle ist ein Referenztyp. Von ihm können Referenzvariablen gebildet werden, die auf Objekte zeigen, deren Klassen die Schnittstelle implementieren.

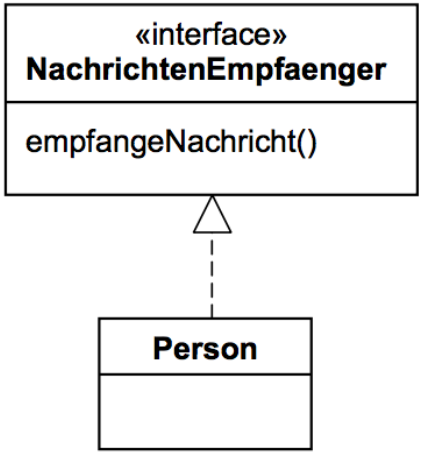
Schnittstellen (Interfaces)

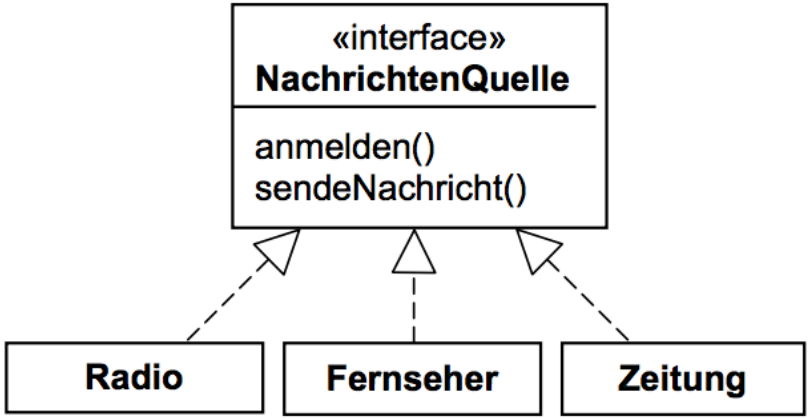
BEISPIEL: SCHNITTSTELLE ALS REFERENZTYP

Kontext



Grafische Darstellung





Definition der Schnittstellen

```
interface NachrichtenQuelle {
public boolean anmelden (NachrichtenEmpfaenger empf);
public void sendeNachricht (String nachricht);
}

interface NachrichtenEmpfaenger {
public void empfangeNachricht (String nachricht);
}
```

Definition der Nachrichtenquelle Zeitung

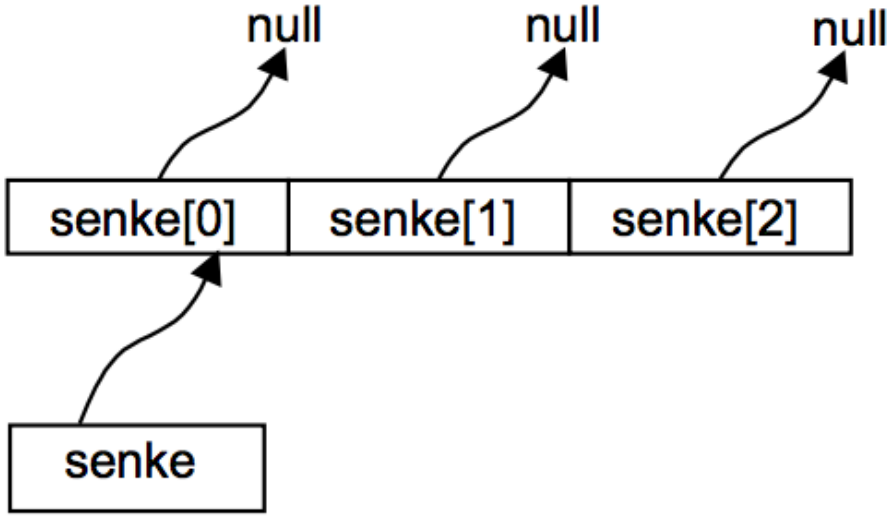
```
public class Zeitung implements NachrichtenQuelle {
   private String name;
   private NachrichtenEmpfaenger[] arr;
   private int anzahlEmpfaenger = 0;

   public Zeitung (String name, int maxAnzahlEmpfaenger){
     this.name = name;
     arr = new NachrichtenEmpfaenger [maxAnzahlEmpfaenger]; }

   public boolean anmelden (NachrichtenEmpfaenger empf) {...}
   public void sendeNachricht (String nachricht) {...}
}
```

Objekterzeugung

```
public class Test {
  public static void main (String[] args) {
    NachrichtenEmpfaenger[] senke = new NachrichtenEmpfaenger[3];
    senke[0] = new Person ("Fischer", "Fritz");
    senke[1] = new Person ("Maier", "Hans");
    senke[2] = new Person ("Kunter", "Max");
    NachrichtenQuelle[] quelle = new NachrichtenQuelle[2];
    quelle[0] = new Zeitung ("-Frankfurter Allgemeine-", 10);
    quelle[0].anmelden (senke [0]);
    quelle[0].anmelden (senke [1]);
    quelle[1] = new Zeitung ("-Südkurier-", 10);
    quelle[1].anmelden (senke [0]);
    quelle[1].anmelden (senke [2]);
  ...
```

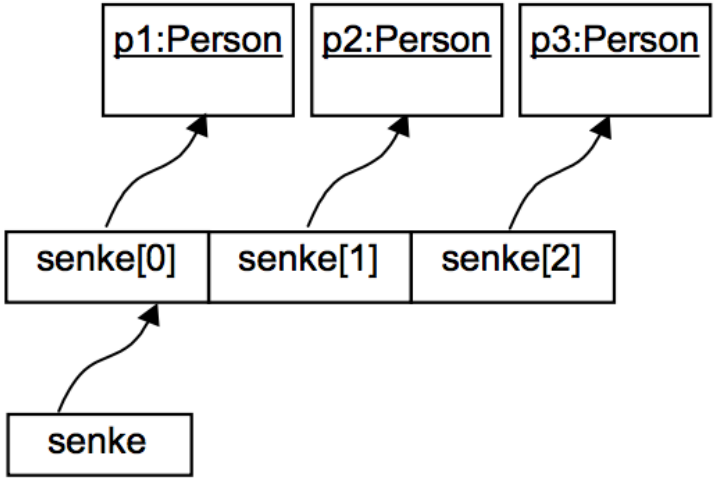
Array von Referenzen eines Schnittstellentyps

```
NachrichtenEmpfaenger[] senke = new NachrichtenEmpfaenger[3];
```



Referenzen im Array zeigen auf konkrete Objekte

```
senke[0] = new Person ("Fischer", "Fritz");
senke[1] = new Person ("Maier", "Hans");
senke[2] = new Person ("Kunter", "Max");
```

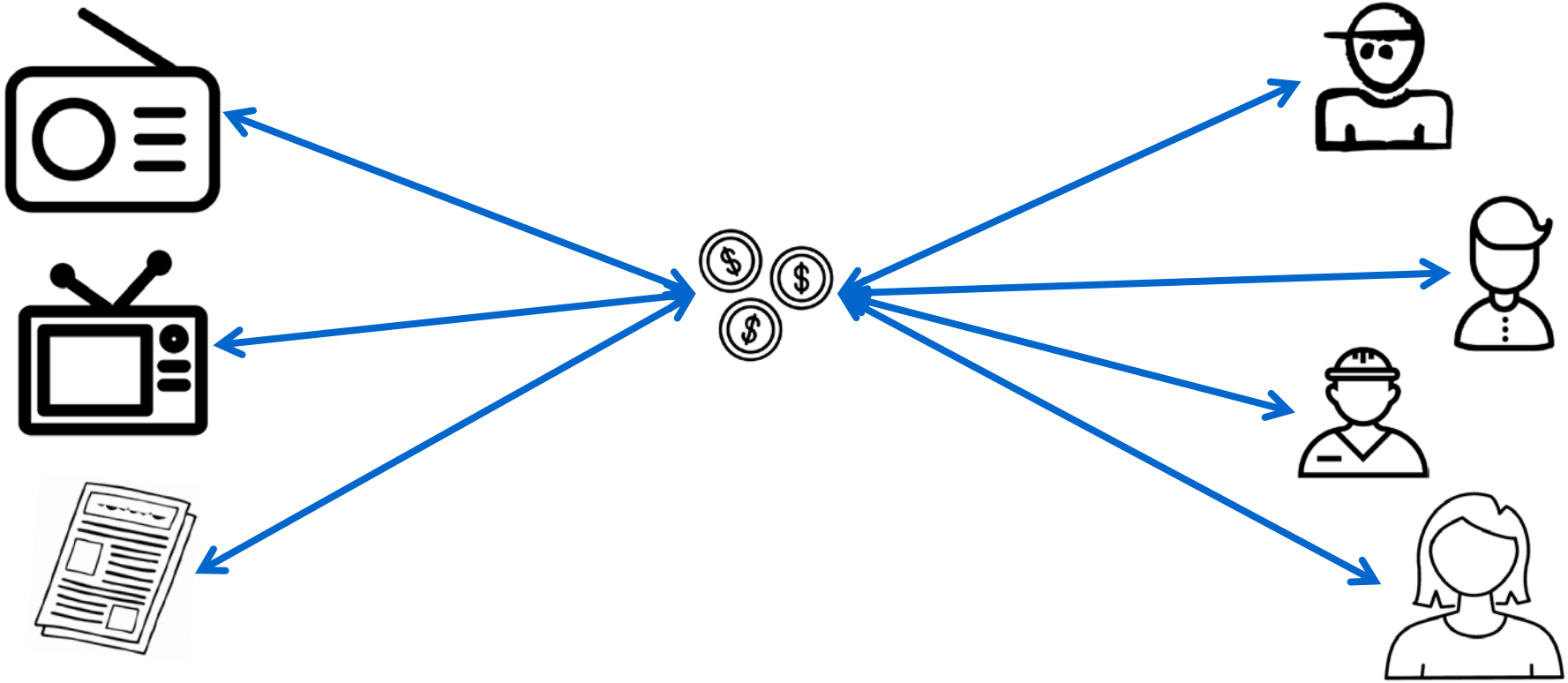


Freie Universität Berlin

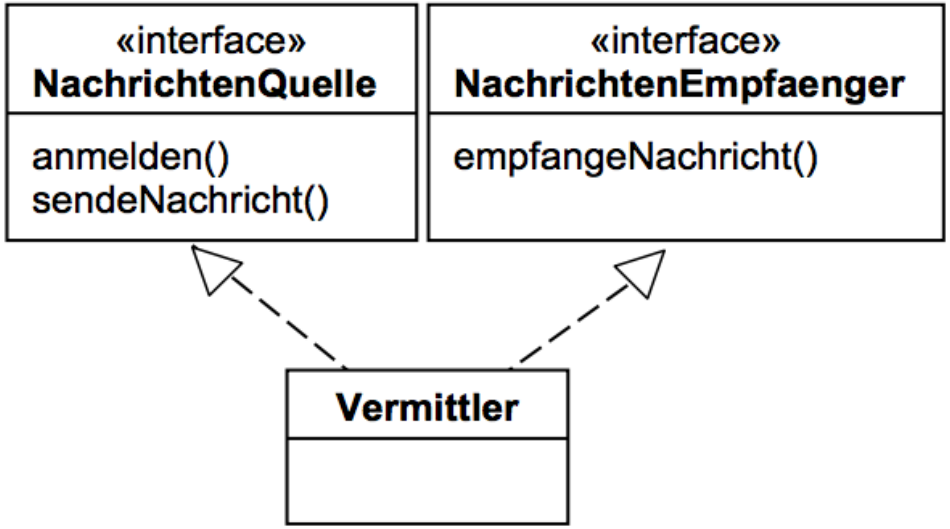
Schnittstellen (Interfaces)

IMPLEMENTIEREN VON MEHREREN SCHNITTSTELLEN

Ergänzter Kontext



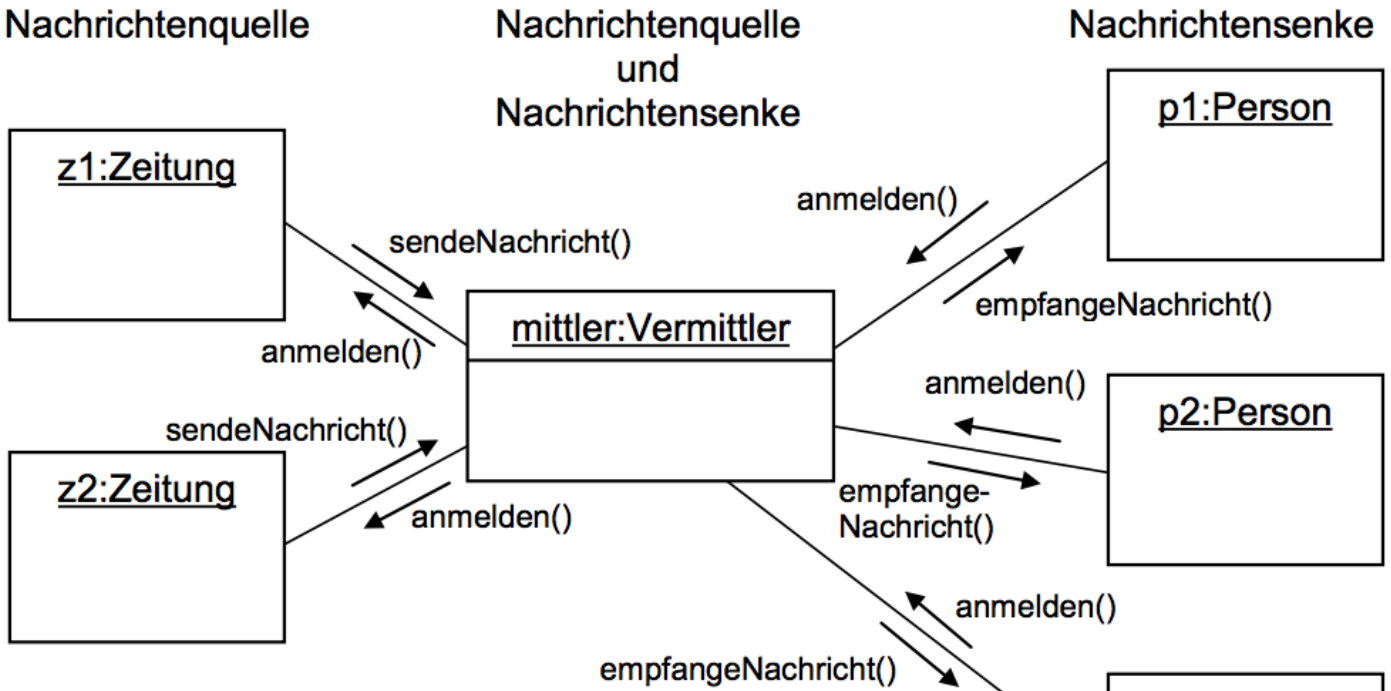
Grafische Darstellung



Die Klasse Vermittler implementiert zwei Schnittstellen

public class Vermittler **implements NachrichtenEmpfaenger, NachrichtenQuelle**

{…}



Ein Objekt der Klasse Vermittler tritt in zweierlei Gestalt auf:
(1) als Objekt vom Typ NachrichtenEmpfaenger und
(2) als Objekt vom Typ NachrichtenQuelle.

Freie Universität Berlin

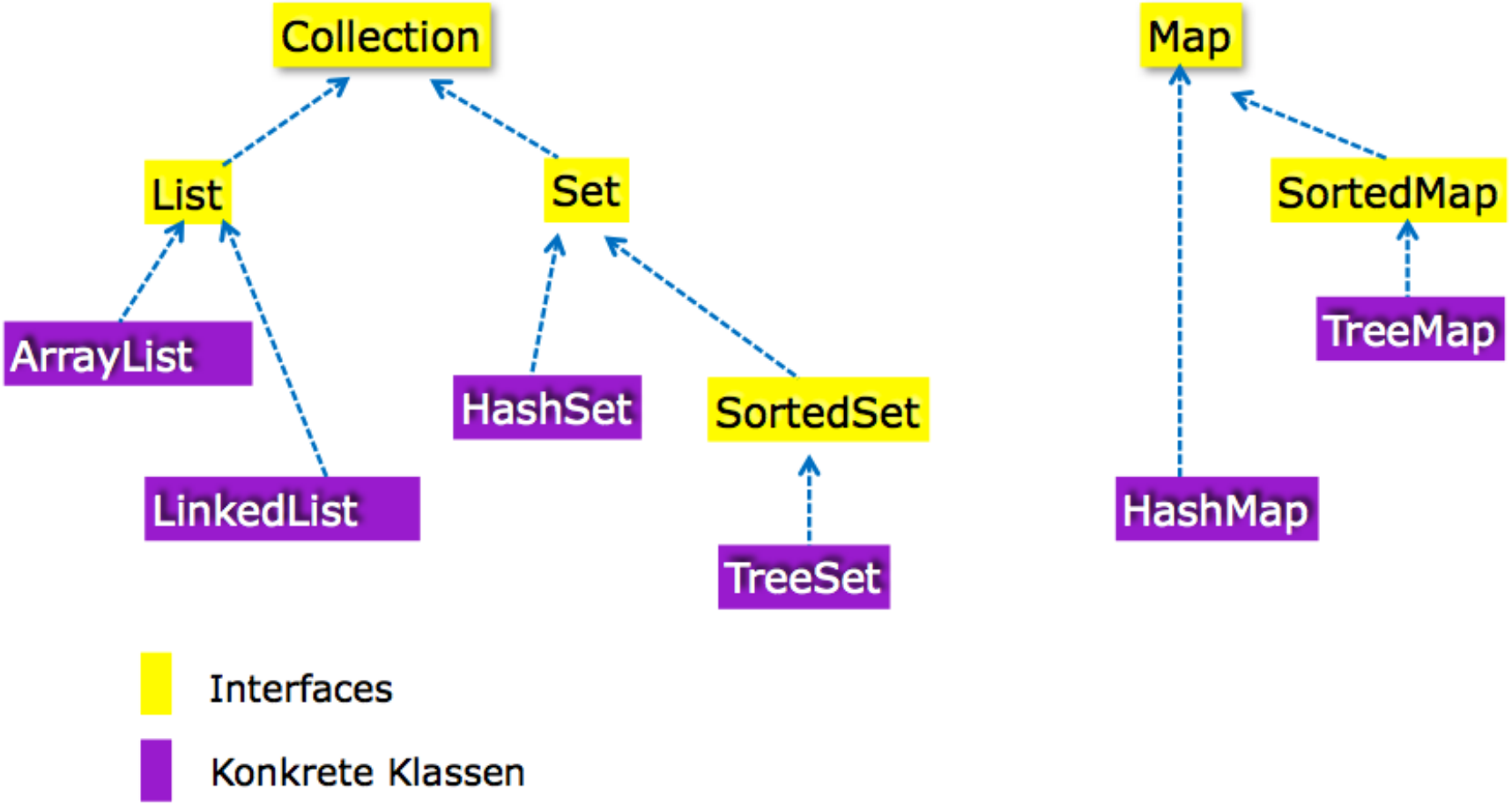
Schnittstellen (Interfaces)

VERERBUNG VON SCHNITTSTELLEN

Unterscheidung bei der Vererbung von Schnittstellen

Einfachvererbung	Mehrfachvererbung
Schnittstellen besitzen – genauso wie Klassen – die Möglichkeit, mit dem Schlüsselwort extends eine schon vorhandene Schnittstelle zu erweitern.	Schnittstellen lassen im Gegensatz zu Klassen Mehrfachvererbung zu.

Collection-Klassen

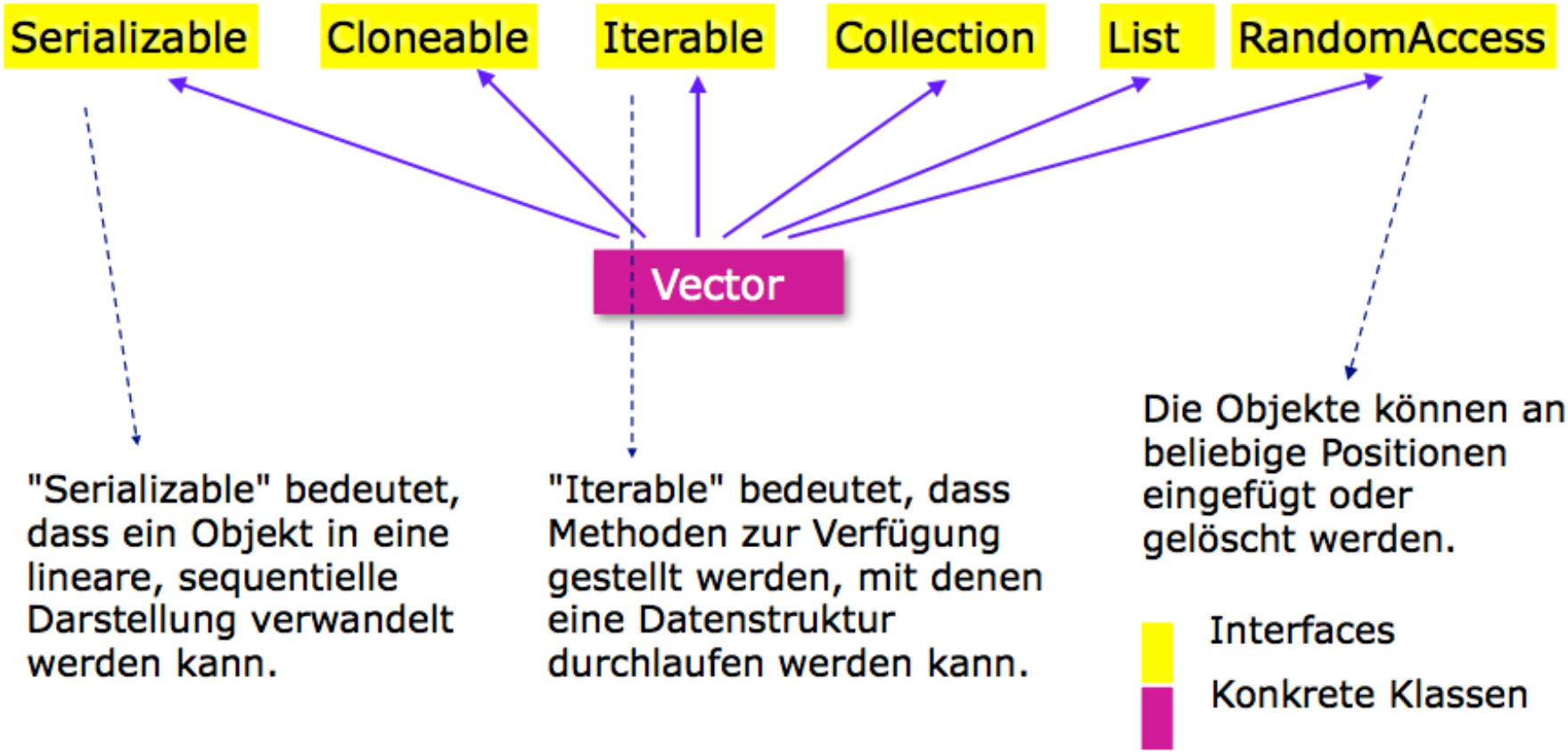


Collections (Sammlungen)

- Die Java Bibliotheken stellen eine Reihe von Schnittstellen und Klassen zur Verfügung, um Sammlungen von Objekten zu simulieren. Es gibt drei Hauptgruppen von "Collections":
- **List**:
  - Die Elementen sind sortiert
  - Duplikate sind erlaubt
  - Die Elemente sind indiziert
- **Set**:
  - Duplikate sind nicht erlaubt
- **Map**:
  - Zuordnung der Elemente zu eindeutigen Schlüsseln
  - Elemente können mehrfach vorkommen

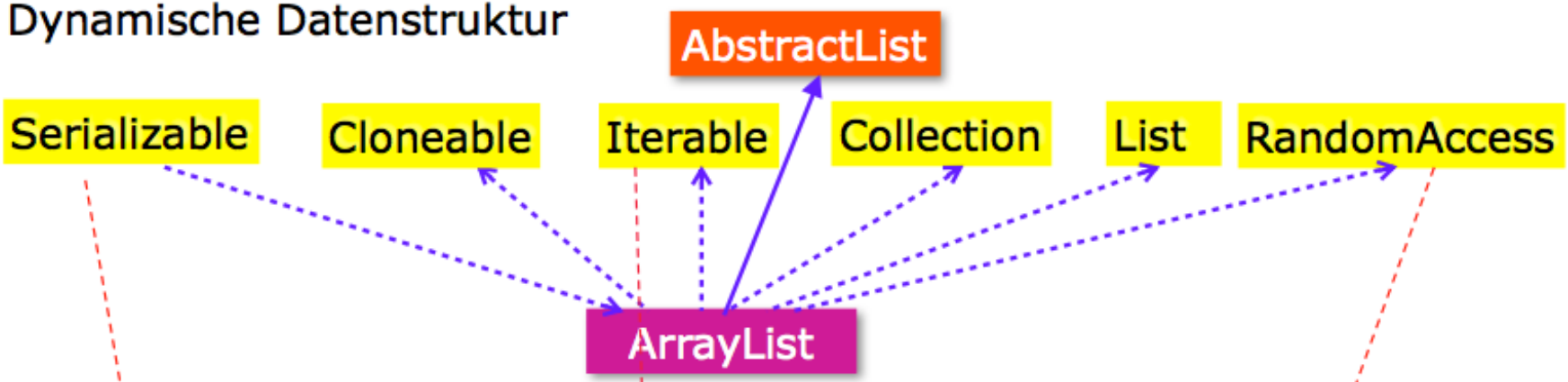
Vektor-Klasse

Dynamische Datenstruktur



ArrayList-Klasse

Dynamische Datenstruktur



"Serializable" bedeutet, dass ein Objekt in eine lineare, sequentielle Darstellung verwandelt werden kann.

"Iterable" bedeutet, dass Methoden zur Verfügung gestellt werden, mit denen eine Datenstruktur durchlaufen werden kann.

Die Objekte können an beliebigen Positionen eingefügt oder gelöscht werden.

Interfaces

Abstrakte Klassen

Konkrete Klassen

Abstrakte Klasse vs. Schnittstelle

- Interfaces sind zwar wie abstrakte Klassen vollständig als abstract deklariert (wenn auch implizit), jedoch enthalten Interfaces ausschließlich abstrakte Methoden und Konstanten.

- Alle Methoden eines Interface sind implizit abstract, und alle Datenelemente sind implizit final static.

- Abstrakte Klassen dagegen können sowohl variable Datenelemente als auch nicht abstrakte, implementierte Methoden besitzen.

Freie Universität Berlin

Schnittstellen und Geschachtelte Klassen

GESCHACHTELTE KLASSEN (INNER CLASSES)

Einführung

- Klassen- oder Schnittstellendefinitionen können zur besseren Strukturierung von Programmen verschachtelt werden.

- Eine Geschachtelte Klasse wird innerhalb des Codeblocks einer anderen Klasse definiert.

- Die bisher eingeführten Klassen werden auch Top-Level-Klassen genannt.

- Einerseits erweisen sich Geschachtelte Klassen als elegante, sehr nützliche Zusätze der Sprache, andererseits gibt es einige Regeln und Sonderfälle, die leicht verwirren können und deshalb vermieden werden sollten.

Geschachtelte Klassen...

- werden insbesondere in Zusammenhang mit der Programmierung von Benutzeroberflächen und Iteratoren eingesetzt,

- können auf Komponenten der sie umfassenden Klasse zugreifen,

- sind außerhalb des sie definierenden Blockes, außer mit Hilfe des voll qualifizierenden Namens, nicht sichtbar.

Ausprägungen von geschachtelten Klassen

Elementklasse

Lokale Klasse

Lokale Klassen mit Klassennamen

Lokale Klassen ohne Klassennamen
(Anonyme Klassen)

Statisch geschachtelte Klasse

Ausprägungen von geschachtelten Klassen

Elementklasse

Lokale Klasse

Lokale Klassen mit Klassennamen

Lokale Klassen ohne Klassennamen
(Anonyme Klassen)

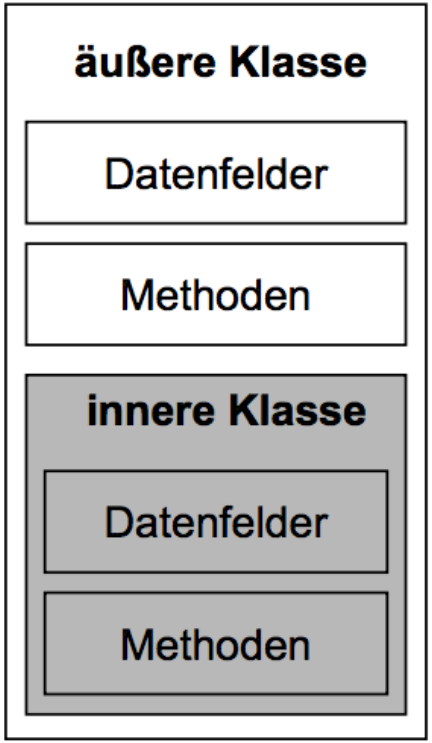
Statisch geschachtelte Klasse

Elementklassen (1)

- Eine Elementklasse ist – wie der Name schon sagt – ein **Element** einer Klasse, das einen Namen trägt.

- **Objekte von Elementklassen**, sog. Elementobjekte, können nur existieren, wenn auch ein Objekt der umschließenden Klasse existiert.

- Eine Methode eines **Elementobjektes** kann auf alle Datenfelder und Methoden des umfassenden Objektes zugreifen (umgekehrt ebenfalls).



Elementklassen (2)

- Da eine Elementklasse ein Element einer Klasse ist, das einen Namen trägt, ist der Zugriffsschutz gleich wie bei den schon bekannten Bestandteilen einer Klasse.

- Elementklassen sind echte innere Klassen im Gegensatz zu den statisch geschachtelten Klassen, die nur zur Strukturierung dienen.

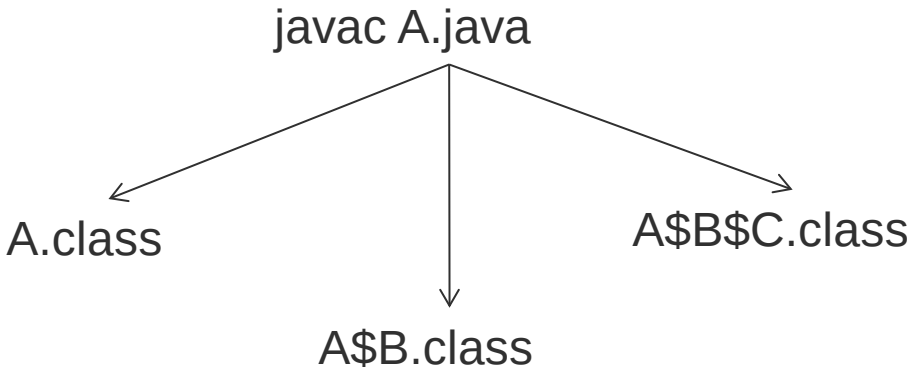
- Elementklassen werden analog gebildet und benutzt wie normale Klassen.

- Der Compiler erzeugt beim Übersetzen einer Klassendatei für alle verschachtelte Klassen und Interfaces eigene .class-Dateien

Übersetzen einer Klassendatei

- Sie tragen den Namen der übergeordneten Klasse(n) und den eigenen, wobei diese mit $ unterteilt werden:

```
public class A {
  . . .
  public class B {
     . . .
     public class C {
        . . .
}}}
```



Syntax einer Elementklasse

```
public class AeussereKlasse {
   .... .
   .... .
   class ElementKlasse {
   .... .
   }
}
```

Erzeugung eines Objekts

```
AuessereKlasse ref = new AuessereKlasse();
AuessereKlasse.ElementKlasse elem = ref.new ElementKlasse();
```

Compiler → `new AuessereKlasse.ElementKlasse (ref);`

Einschränkungen für Elementklassen

- Elementklassen dürfen keine Klassenvariablen, Klassenmethoden oder statische Klassen beinhalten. Klassenvariablen und Klassenmethoden sind nur bei einer statisch geschachtelten Klasse erlaubt.

- Elementklassen dürfen nicht den gleichen Namen wie eine umschließende Klasse besitzen.

Ausprägungen von geschachtelten Klassen

Elementklasse

Lokale Klasse

Lokale Klassen mit Klassennamen

Lokale Klassen ohne Klassennamen
(Anonyme Klassen)

Statisch geschachtelte Klasse

Lokale Klassen

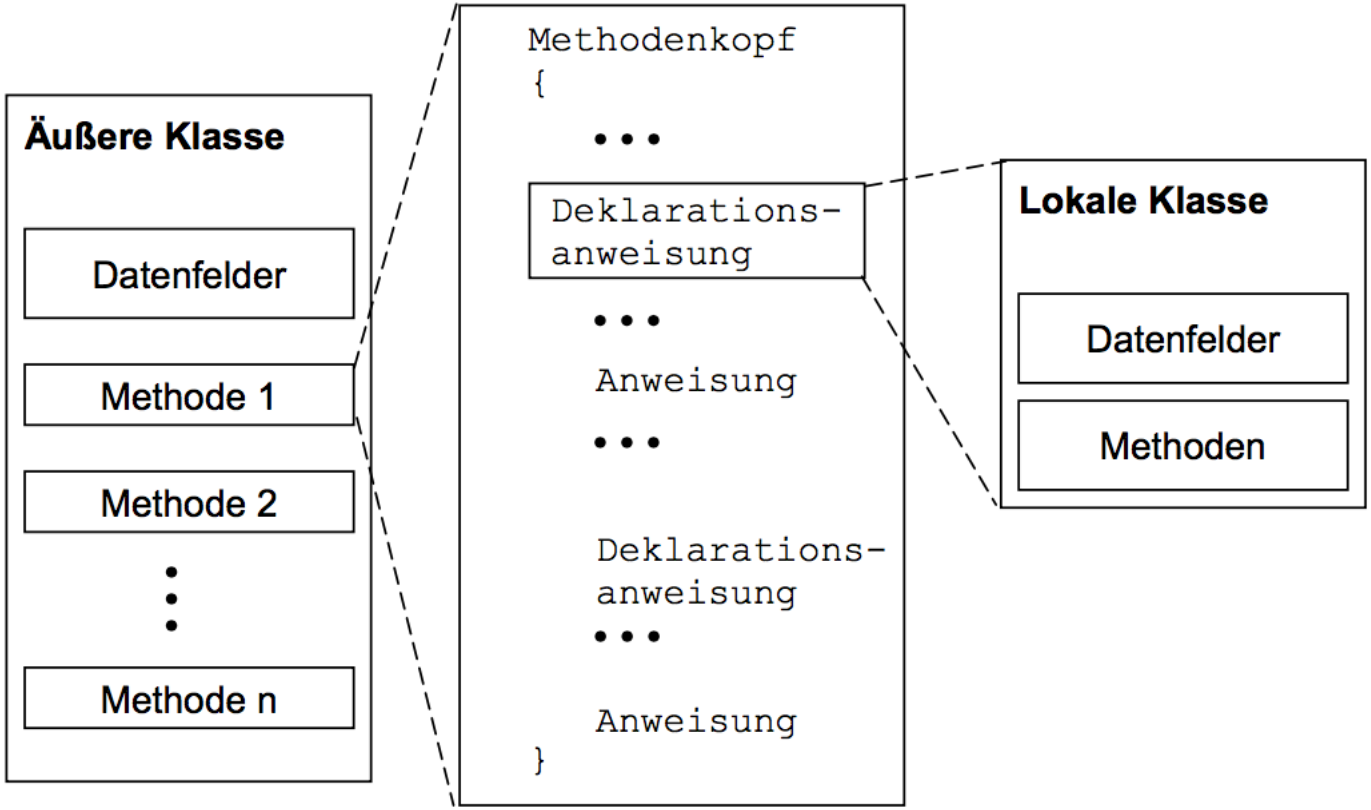
- Lokale Klassen sind innere Klassen, die nicht auf oberer Ebene in anderen Klassen verwendbar sind, sondern nur lokal innerhalb von Anweisungsblöcken von Methoden.

- Lokale Klassen
  - dürfen nicht als public, protected, private oder static deklariert werden.
  - dürfen keine statischen Elemente haben (Felder, Methoden oder Klassen).
  - dürfen nicht den gleichen Namen tragen wie eine der sie umgebenden Klassen.
  - dürfen im umgebenden Codeblock nur die mit final markierten Variablen und Parameter benutzen.

Ausprägungen von geschachtelten Klassen

- Elementklasse
- Lokale Klasse
  - Lokale Klassen mit Klassennamen
  - Lokale Klassen ohne Klassennamen (Anonyme Klassen)
- Statisch geschachtelte Klasse

Lokale Klassen mit Klassennamen



Ausprägungen von geschachtelten Klassen

- Elementklasse
- Lokale Klasse
  - Lokale Klassen mit Klassennamen
  - Lokale Klassen ohne Klassennamen (Anonyme Klassen)
- Statisch geschachtelte Klasse

Anonyme Klasse

- Eine anonyme Klasse ist eine lokale Klasse, deren Klassennamen für den Programmierer nicht sichtbar und damit auch nicht verwendbar ist.

- Anonyme Klassen werden wie lokale Klassen innerhalb von Anweisungsblöcken definiert haben.

- Sie entstehen immer zusammen mit einem Objekt, sie haben keinen Konstruktor und sie haben die gleichen Beschränkungen wie Lokale Klassen.

Syntax für Anonyme Klassen

```
...
public void doSomething() {
   ...
   ITypen refTyp = new ITypen()
   {
      public boolean testTyp(){...}
   }
    // Zugriff auf die Methode der anonymen Klasse
    if (refTyp.testTyp())
    ...
```

Übersetzung des Compilers

```
class Viewer$1 implements ITypen
{
    //Datenfelder und Methoden der Klasse
}
ITypen refTyp = new Viewer$1();
```

Einschränkungen für anonyme Klassen

- Anonyme Klassen dürfen keine als static deklarierten Datenfelder, Methoden oder Klassen definieren.

- Anonyme Klassen können keinen Konstruktor haben, da sie auch keinen Namen besitzen.

- Von anonymen Klassen kann nur eine einzige Instanz erzeugt werden. Deshalb sollte man lokale Klassen mit Klassennamen oder Elementklassen den anonymen Klassen dort vorziehen, wo man mehrere Instanzen einer inneren Klasse benötigt.

Ausprägungen von geschachtelten Klassen

Elementklasse

Lokale Klasse

Lokale Klassen mit Klassennamen

Lokale Klassen ohne Klassennamen
(Anonyme Klassen)

Statisch geschachtelte Klasse

Beschreibung

- Statisch geschachtelte Klassen und statisch geschachtelte Schnittstellen werden auch als sogenannte statische Top-Level-Klassen bzw. statische Top-Level-Schnittstellen bezeichnet.

- Bei statischen Top-Level-Elementen braucht man kein Objekt einer äußeren Klasse, um ein Objekt einer inneren Klasse zu erzeugen.

- Das Schlüsselwort static hat zur Konsequenz, dass diese Klasse bzw. Schnittstelle sich vollkommen gleich wie jede andere normale Top-Level-Klasse /Schnittstelle verhält, mit dem Unterschied, dass die geschachtelte Klasse über den Namen der umschließenden Klasse angesprochen wird.

Einfaches Beispiel

```
class EnclosingClass{
. . .
    static class StaticNestedClass {
    . . .
    }
    class InnerClass {
    . . .
    }
} // end of class
```

Geschachtelte Top-Level-Klassen und Interfaces verhalten sich wie andere Paket-Elemente auf der äußersten Ebene, außer dass ihrem Namen der Name der umgebenden Klasse vorangestellt wird. Sie werden als static deklariert.

Strukturierung des Namenraums von Klassen

```
public class A {
  static int i = 4711;
  public static class B {
    int my_i = i;
      public static class C { ... }//end..C
  }// end of class B
}// end of class A

//Objekterzeugung
A a = new A();
A.B ab = new A.B();
A.B.C abc = new A.B.C();
```