Barry Linnert
Institut für Informatik

Freie Universität Berlin

Programmieren

Barry Linnert

Sommersemester 2020

Gliederung der heutigen Vorlesung

- Kurze Wiederholung
- Motivation – warum ist Vererbung sinnvoll?
  - Modellierung von Klassenhierarchierung (Spezialisierung/Generalisierung)
  - Einfache und mehrfache Vererbung
- Einschub: Überladen von Methoden
- Vererbung in Java
- Weitere objektorientierte Konzepte in Java
  - Abstrakte Klassen und Methoden
  - Interfaces (Schnittstelle)
- Zusammenfassung

Freie Universität Berlin

Vererbung (inheritance)

WIEDERHOLUNG

Objekterzeugung

- Objekte werden durch den Aufruf von Konstruktoren erzeugt. Ein Konstruktor wird mit Hilfe der **new**-Operatoren aufgerufen. Zum Beispiel:

  Rectangle r1 = new Rectangle();

- Eine Klassendefinition kann mehrere Konstruktoren haben mit verschiedenen Initialisierungen der Objekteigenschaften.

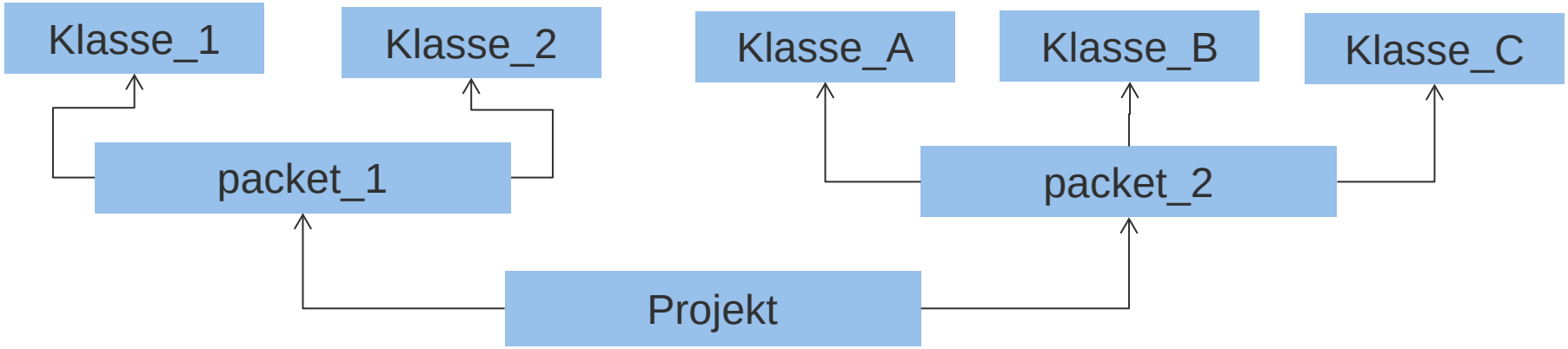
- Wenn in einer Klasse keine Konstruktoren definiert worden sind, werden die Eigenschaften von Objekten mit Defaultwerten initialisiert.

Konstruktoren

- Ein guter OOP-Stil bedeutet, geeignete *Konstruktoren* zu definieren, die Objekte initialisieren und evtl. initiale Berechnungen durchführen.

- Syntaktisch sind Konstruktoren Methoden mit einigen speziellen Eigenschaften:
  - ihr Name muss mit dem Namen der Klasse übereinstimmen,
  - sie dürfen keinen Ergebnistyp, auch nicht void, haben,
  - sie werden mit dem new-Operator aufgerufen werden und erzeugen ein Objekt der Klasse.

Ebenen Modualisierung




```
package packet_1 ;
public class Klasse_1 {
.....
}
```

Sichtbarkeit von Java Variablen

Zugriffsangabe oder Sichtbarkeit	Klasse	Unterklasse	Paket	Welt
private	X			
kein Modifizierer (package)	X		X	
protected	X	X	X	
public	X	X	X	X

Freie Universität Berlin

Vererbung (Inheritance)

MOTIVATION

```
public class Rectangle {

    // Attribute
    int x;
    int y;
    int width;
    int height;


    // Methoden
    public int perimeter() {
    return 2*(width + height);
    }

    public int area() {
    return (width * height);
    }

} // end of class Rectangle
```

Eigenschaften

Operationen

Berechne Fläche
Berechne Umfang

```
public class Rechteck {

   private int xPos;
   private int yPos;
   private int breite;
   private int hoehe;

   public Rechteck(int x, int y,
                   int breite, int hoehe) {

      setX(x);
      setY(y);
      setBreite(breite);
      setHoehe(hoehe);
   }

   public void setX(int x) {
      if (x >= 0) xPos = x;
   }
```

```
public class Kreis {

   private int xPos;
   private int yPos;
   private int radius;

   public Kreis(int x, int y,
                int radius) {

      setX(x);
      setY(y);
      setRadius(radius);
   }

   public void setX(int x) {
      if (x >= 0) xPos = x;
   }
```

```
public int getX() {
   return xPos;
}

public void setY(int y) {
   if (y >= 0) yPos = y;
}

public int getY() {
   return yPos;
}

public void setBreite(
              int breite) {
   if (breite > 0)
      this.breite = breite;
}

public int getBreite() {
    return breite;
}
```

```
public int getX() {
   return xPos;
}

public void setY(int y) {
   if (y >= 0) yPos = y;
}

public int getY() {
   return yPos;
}

public void setRadius(
              int radius) {
   if (radius > 0)
      this.radius = radius;
}

public int getRadius() {
   return radius;
}
```

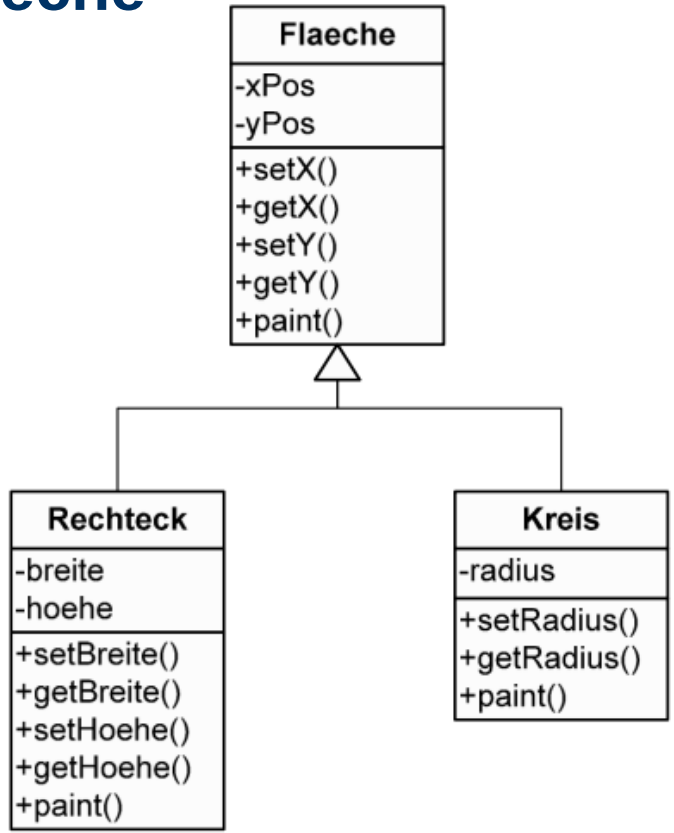
```
public void setHoehe(
                int hoehe) {
    if (hoehe > 0)
        this.hoehe = hoehe;
}

public int getHoehe() {
    return hoehe;
}
```

```
public void paint() {
    System.out.println("------");
    System.out.println("|    |");
    System.out.println("------");
}
}
```

```
public void paint() {
    System.out.println("/---\\");
    System.out.println("|    |");
    System.out.println("\\---/");
}
}
```

Definition der Oberklasse Flaeche



Motivation

- Ein wesentliches Merkmal objektorientierter Sprachen ist die Zusammenfassung von Variablen und Methoden zu Klassen.
- Hauptziel von objektorientierten Programmierkonzepten ist es, die Flexibilität, leichte Anpassbarkeit und Wiederverwendbarkeit von Software zu vereinfachen.
- Ein wesentliches Merkmal objektorientierter Sprachen ist die Möglichkeit, Eigenschaften vorhandener Klassen auf neue Klassen zu übertragen (Wiederverwendbarkeit).
- Vererbung ist das Konzept der objektorientierten Programmierung und deshalb in allen objektorientierten Sprachen enthalten.

Was ist Vererbung?

- Der Hauptzweck der Vererbung liegt in der Nutzung der Ähnlichkeit von neu zu schaffenden Klassen zu vorhandenen Klassen und zwar im Sinne von

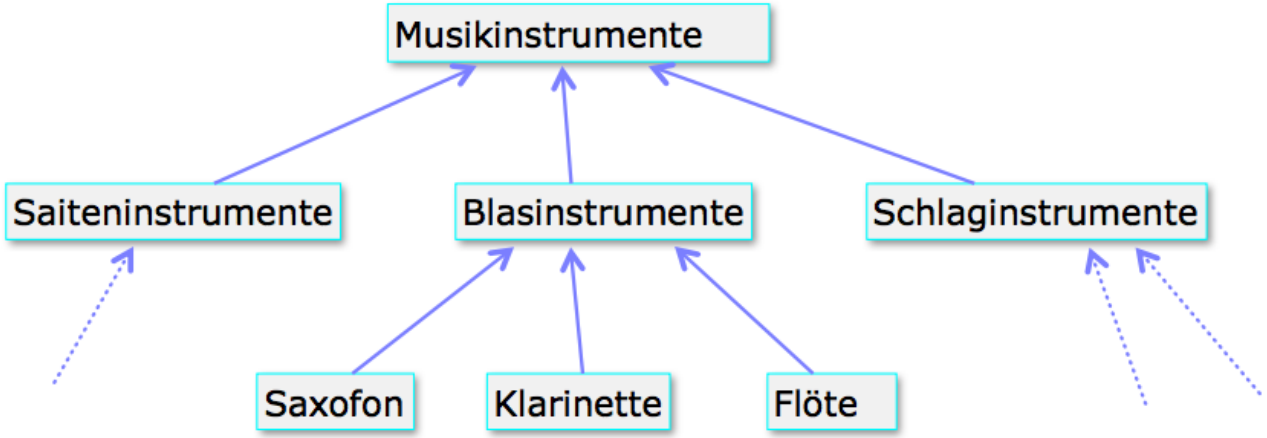
- **Spezialisierung**
  - Erweiterung - spezifische Attribute und Methoden legt man in einer Subklasse an und

- **Generalisierung**
  - Abstraktion - allgemeingültige Attribute und Methoden gehören in eine Superklasse.

Modellierung von Klassenhierarchien

- Bei guten Modellierungen klingt es logisch, wenn gesagt wird, dass ein Element der Unterklasse auch ein Element aller ihrer Oberklassen ist.

- Beispiel



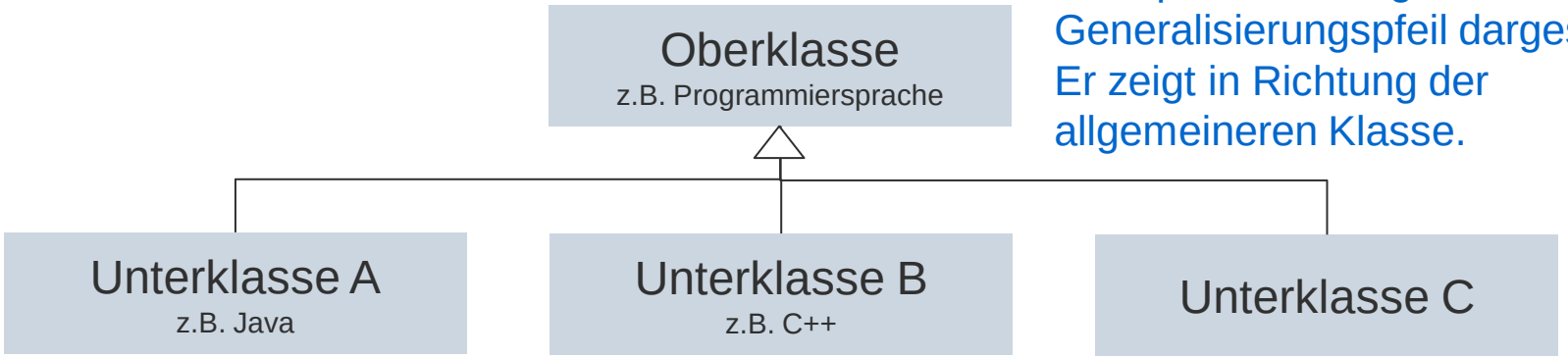
Modellierung von Hierarchien

- Die Kunst ist es, eine möglichst gute Klassenhierarchie für die Modellierung von Softwaresystemen zu finden.

- Nicht alle Teile eines Problems können gut mit rein objektorientierten Techniken gelöst werden.

- Nicht immer gelingt es, eine saubere Klassenhierarchie zu finden!

Klassenhierarchie



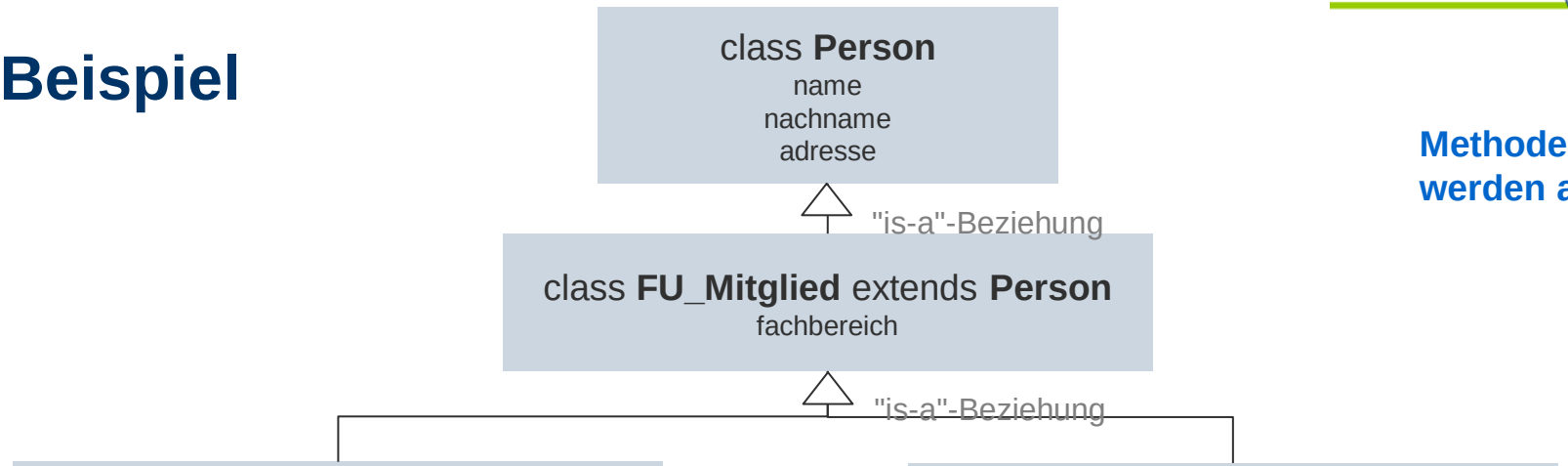
Die Spezialisierung wird als Generalisierungspfeil dargestellt. Er zeigt in Richtung der allgemeineren Klasse.

Die Unterklasse A besitzt alle Eigenschaften und Methoden ihrer Oberklasse

+

die Erweiterungen, die in der Unterklasse A selber definiert worden sind.

Beispiel



Methoden werden auch vererbt.

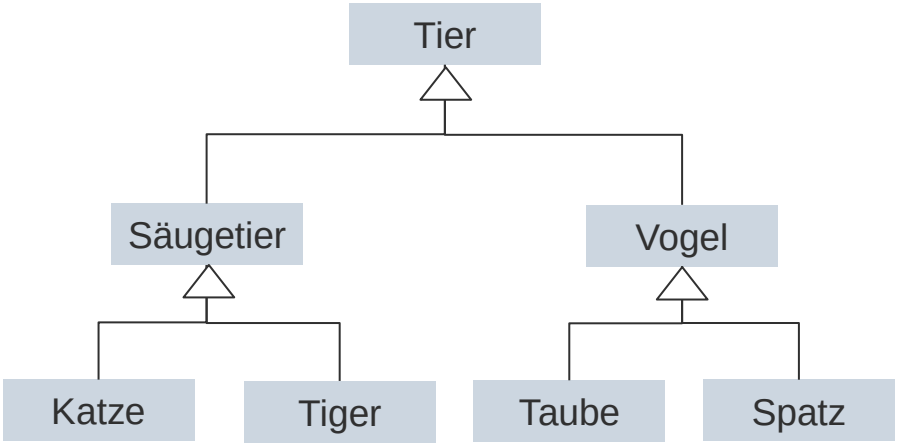
Dozent
name
nachname
addresse
fachbereich
fachrichtung
telefon
büronummer

Student
name
nachname
addresse
fachbereich
Matrikelnummer
semester

Einfache Vererbung

- Eine einfache Vererbung liegt dann vor, wenn die Vererbungshierarchie ein Baum ist, d.h. außer der Basisklasse (Wurzel) hat jede Klasse genau eine direkte Oberklasse.

Beispiel



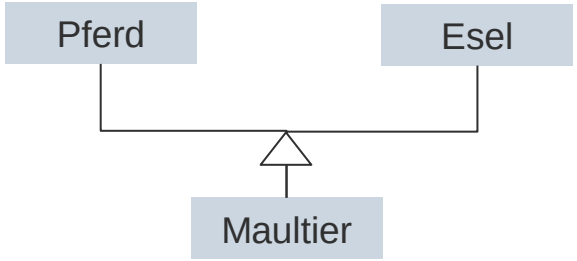
Java und C#

Mehrfache Vererbung

- Bei mehrfacher Vererbung hat mindestens eine Klasse mehr als eine direkte Oberklasse. Durch diese Eigenschaft ist es möglich, voneinander unabhängige Klassenhierarchien zusammenzuführen.

- Bei mehrfacher Vererbung besteht die Gefahr der Namenskollisionen, weil Methodennamen oder Attribute mit gleichem Namen aus verschiedenen Oberklassen vererbt werden können.

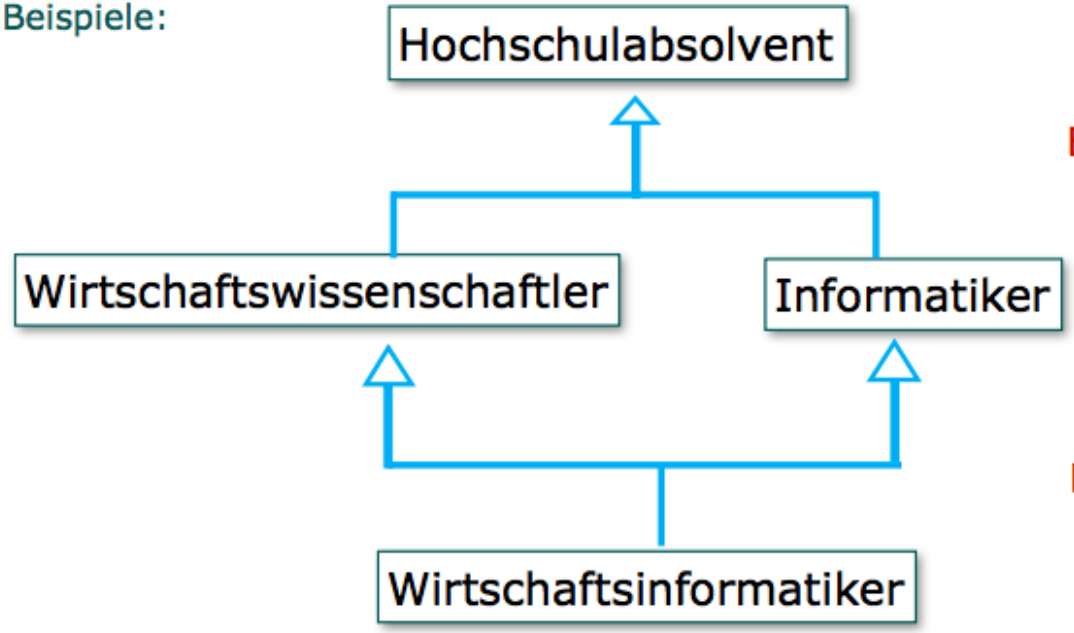
Beispiel



C++ und Python

Beispiel 2 für einfache und mehrfache Vererbung

Beispiele:



Einfache Vererbung

Mehrfache Vererbung

Vererbung (Inheritance)

EINSCHUB

Überladen von Konstruktoren

- Die Konstruktoren werden vom Übersetzer durch die Anzahl und den Typ der Parameter unterschieden.

```
public Person ( String vorname, String nachname, Date geburtsdatum )
public Person ( String vorname, String nachname )
public Person ( String vorname )
public Person ()
```

Signatur eines Konstruktors

Überladen von Methoden

- Die Methoden werden vom Übersetzer durch Anzahl und Typ der Parameter unterschieden.

```
public void draw ( String s )
public void draw ( int i )
public void draw ( double d )
```

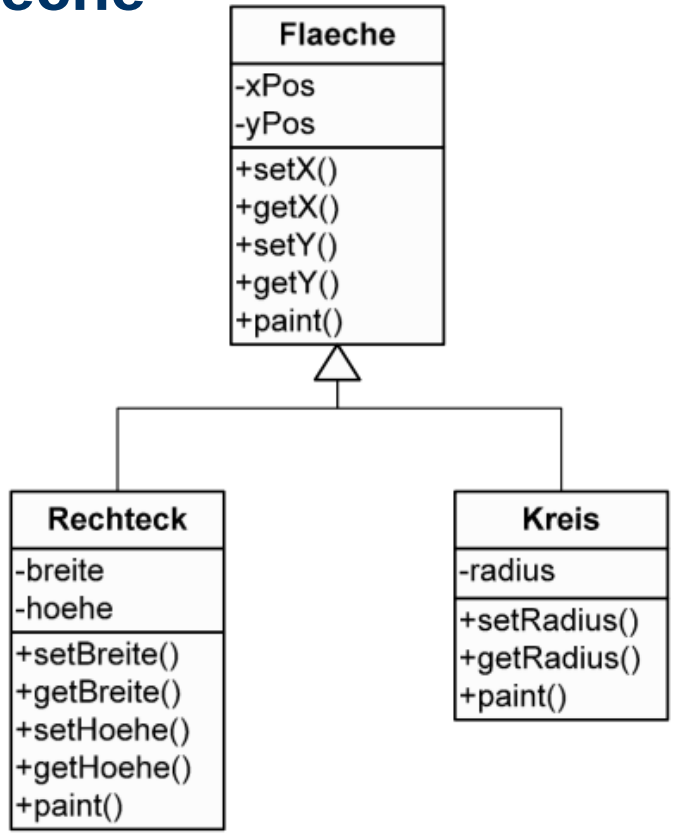
- Die Parametertypen bestimmen die Signatur einer Methode.

Freie Universität Berlin

Vererbung (Inheritance)

VERERBUNG IN JAVA

Definition der Oberklasse Flaeche

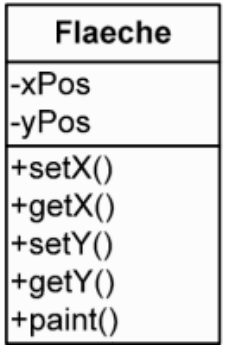


```
public class Flaeche {

  // Variablen
  private int xPos;
  private int yPos;

  // Konstruktor
  public Flaeche(int x, int y) { setX(x); setY(y); }

  // Methoden
  public void setX(int x) { if (x >= 0) xPos = x; }
  public int getX() { return xPos; }
  public void setY(int y) { if (y >= 0) yPos = y; }
  public int getY() { return yPos; }
  public void paint() { System.out.println("XXX"); }
}
```



```
public class Rechteck
 extends Flaeche {

 private int breite;
 private int hoehe;

 public Rechteck(int x, int y,
         int breite, int hoehe) {

    super(x, y);
    setBreite(breite);
    setHoehe(hoehe);
 }

 public void setBreite(
              int breite) {
    if (breite > 0)
       this.breite = breite;
 }
```

```
public class Kreis
 extends Flaeche {

private int radius;

 public Kreis(int x, int y,
              int radius) {

    super(x, y);
    setRadius(radius);
 }

 public void setRadius(
              int radius) {
    if (radius > 0)
       this.radius = radius;
 }
```

```
public int getBreite() {
    return breite;
}

public void setHoehe(
             int hoehe) {
   if (hoehe > 0)
      this.hoehe = hoehe;
}

public int getHoehe() {
    return hoehe;
}
public void paint() {
   System.out.println("------");
   System.out.println("|    |");
   System.out.println("------");
}
}
```

```
public int getRadius() {
   return radius;
}

public void paint() {
   System.out.println("/---\\");
   System.out.println("|   |");
   System.out.println("\\---/");
}
}
```

Konstruktoren in Unterklassen

- **Konstruktoren werden nicht vererbt**, d.h. Unterklassen müssen jeweils eigene Konstruktoren angeben und die Oberklassenkonstruktoren explizit aufrufen.

- Wenn man keinen Konstruktor der Oberklasse verwendet, wird *nur* der implizite default-Konstruktor **super**() aufgerufen.

- Wenn man einen Konstruktor der Oberklasse verwenden will, muss der **Aufruf am Anfang** der jeweiligen Konstruktoren stehen.

- Das Schlüsselwort **super** dient nicht nur dazu, um Konstruktoren der Oberklasse aufzurufen, sondern wird verwendet, um den Zugriff auf verdeckte Instanzvariablen und Methoden der Oberklasse zu ermöglichen.

Aufruf der Konstruktors der Oberklasse

```
public class Circle extends Area {
  double x, y, r;
  public Circle(){
  }
  ...
}
```

Der Konstruktor der Oberklasse wird implizit aufgerufen.

```
public class Circle extends Area {
  double x, y, r;
  public Circle(){
    super(x,y);
  }
  ...
}
```

Der Konstruktor der Oberklasse wird explizit aufgerufen.

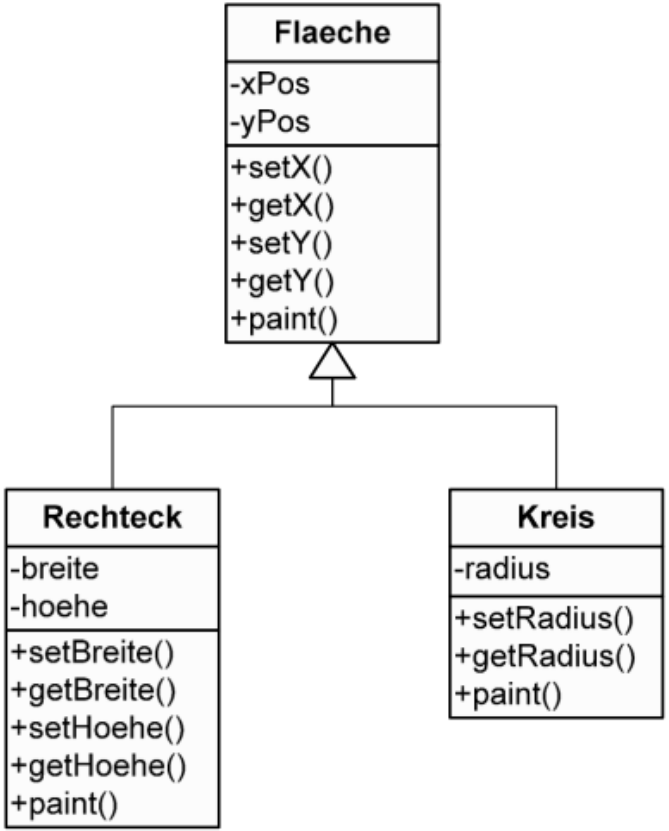
- Wenn man keinen Konstruktor der Oberklasse verwendet, wird *nur* der implizite default-Konstruktor **super**() aufgerufen.

Beispiel 1

```
public class Person {
   String vorname;
   String nachname;
   public Person ( String vorname, String nachname ) {
      this.vorname = vorname;
      this.nachname = nachname;
   }
}
```

```
public class Student extends Person {
   String fachbereich;
   public Student ( String vorname, String nachname, String fachbereich ) {
      super(vorname, nachname);
      this.fachbereich = fachbereich;
   }
}
```

Vererbung von Methoden



Überschreiben von Methoden

- Der aktuelle Typ des aufgerufenen Objekts bestimmt, welche Methode tatsächlich benutzt wird.

```
...
Flaeche figure_1 = new Rechteck(0,0,10,10);
Flaeche figure_2 = new Kreis(0.0,0.0,1.0);
...
figure_1.paint();
figure_2.paint();
...
```

Verhindern des Überschreibens von Methoden

- Eine als **final** markierte Methode kann nicht in Unterklassen überschrieben werden
  public final String nachname()

- Von einer **final** Klasse können keine Unterklassen gebildet werden
  public final class String { ... }

Überschreiben von Feldern

- Es gibt bei Instanzvariablen keine dynamische Bindung, weil Instanzvariablen im Gegensatz zu Methoden einen anderen Typ haben können als in der Superklasse.

- Zugriff auf Instanzvariablen mit dem gleichen Namen in der Oberklasse erfolgt nur mit **super.variablenname**

Beispiel: Überschreiben von Feldern

```
public class Person {
  ...
  public String name() {
     return name;
  }
}
```

```
public class Mann extends Person {
   ...
   public String name() {
      return "Herr "+ super.name();
   }
}
```

Die Instanzvariable **name** in der **Mann-Klasse** verdeckt die Instanzvariable **name** in **Person**.

Beispiel: Überschreiben von Methoden und Feldern

```
public class O {
   int y;
   int x = 10;
   public int q() {
        return x*x;
   }
}
```

```
public class U extends O {

int x = 2;

int q() { return x * x; }
int q1() { return super.x * super.x; }
int q2() { return super.q(); }
}
```

Verdeckt das x der Oberklasse

Verdeckt die q-Methode der Oberklasse

```
...
U u = new U();
System.out.println( u.q1() );
System.out.println( u.q() );
System.out.println( u.q2() );
...
```

Ausgabe

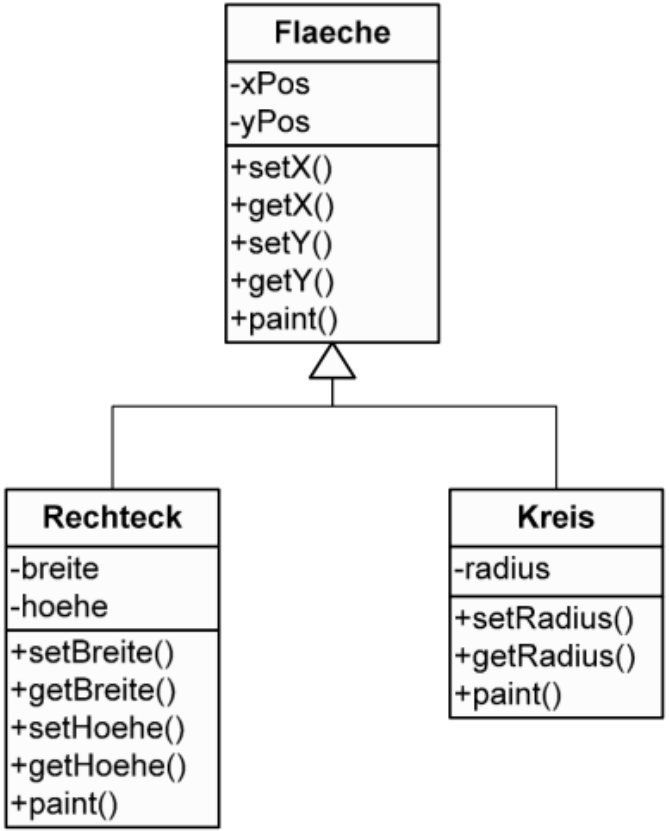
```
100
4
100
```

Freie Universität Berlin

Vererbung (Inheritance)

ABSTRAKTE KLASSEN UND METHODEN

Was ist das Objekt Flaeche?

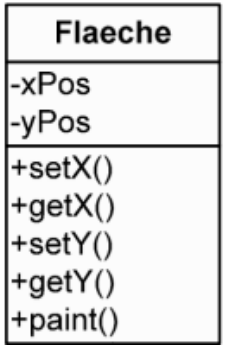


```
public abstract class Flaeche {

  // Variablen
  private int xPos;
  private int yPos;

  // Konstruktor
  public Flaeche(int x, int y) { setX(x); setY(y); }

  // Methoden
  public void setX(int x) { if (x >= 0) xPos = x; }
  public int getX() { return xPos; }
  public void setY(int y) { if (y >= 0) yPos = y; }
  public int getY() { return yPos; }
  public abstract void paint();
}
```



Abstrakte Methoden

- Abstrakte Methoden enthalten **nur die Deklaration des Methodenkopfes**, aber **keine Implementierung** des Methodenrumpfes.

- Abstrakte Methoden haben anstelle der geschweiften Klammern mit den auszuführenden Anweisungen ein Semikolon am Ende. Zusätzlich wird die Definition mit dem Attribut **abstract** versehen.

- Abstrakte Methoden definieren nur eine Schnittstelle, die durch Überschreiben innerhalb einer abgeleiteten Klasse implementiert werden kann.

Abstrakte Klassen

- Eine Klasse, die mindestens eine abstrakte Methode enthält, muss als abstrakte Klasse deklariert werden.

- Die bereits implementierten Methoden in einer abstrakten Klasse können von anderen Klassen geerbt werden, welche die abstrakten Methoden zusätzlich implementieren können.

- **Eine abstrakte Klasse wird in einer Unterklasse konkretisiert, wenn dort alle ihre abstrakten Methoden implementiert sind.**

- Abstrakte Klassen werden mit dem Schlüsselwort **abstract** markiert.

Abstrakte Klassen

- Abstrakte Klassen sind künstliche Oberklassen, die geschaffen werden, um Gemeinsamkeiten mehrerer Klassen zusammenzufassen.
- Abstrakte Klassen dienen nur zur besseren Strukturierung der Software.
- Objekte können nicht aus einer abstrakten Klasse erzeugt werden.
- Die fehlende Implementierung wird in den Unterklassen "nachgeliefert", sonst sind diese auch abstrakt.

Vererbung (inheritance)

INTERFACE

Was sind Interfaces?

- 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.

- Ein Interface ist wie eine abstrakte Klasse, die ausschließlich abstrakte Methoden besitzt.

Interfaces

- Interfaces sind vollständig abstrakte Klassen.
- Alle Methoden sind implizit **abstract** und **public**, aber keine ist implementiert.
- Keine Instanzvariable ist deklariert.
- Beispiel

```
public interface I {
  void a();
  void b();
}
```

```
public class C implements I {
  public void a() {
    // Implementierung von a()... }
  public void b() {
    // Implementierung von b()... }
}
```

Vorteile von Interfaces

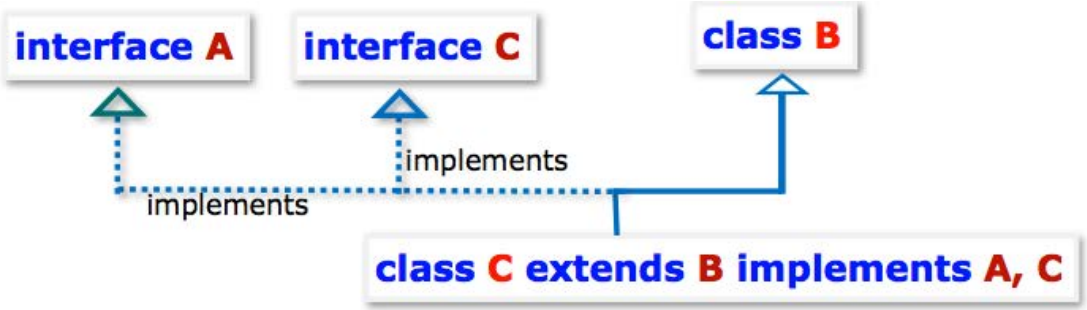
- Verschiedene Implementierungen desselben Typs sind möglich und die Implementierungen können geändert werden, ohne dass der Benutzer es merkt!

```
public class Set implements Collection {
   public void add( Object o ) { ... }
   public void remove( Object o ) { ... }
   public boolean contains( Object o ) {... }
}
```

Alle Methoden des Interface müssen implementiert werden.

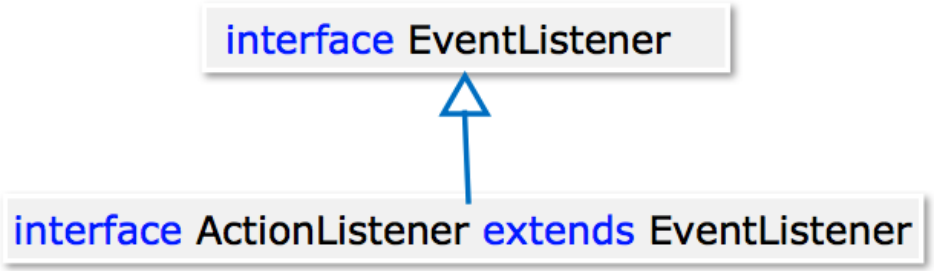
Interfaces als Strukturierungsmittel

- 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.
- Eine Klasse im Java kann von einer Oberklasse vererben und gleichseitig mehrere Interfaces implementieren.

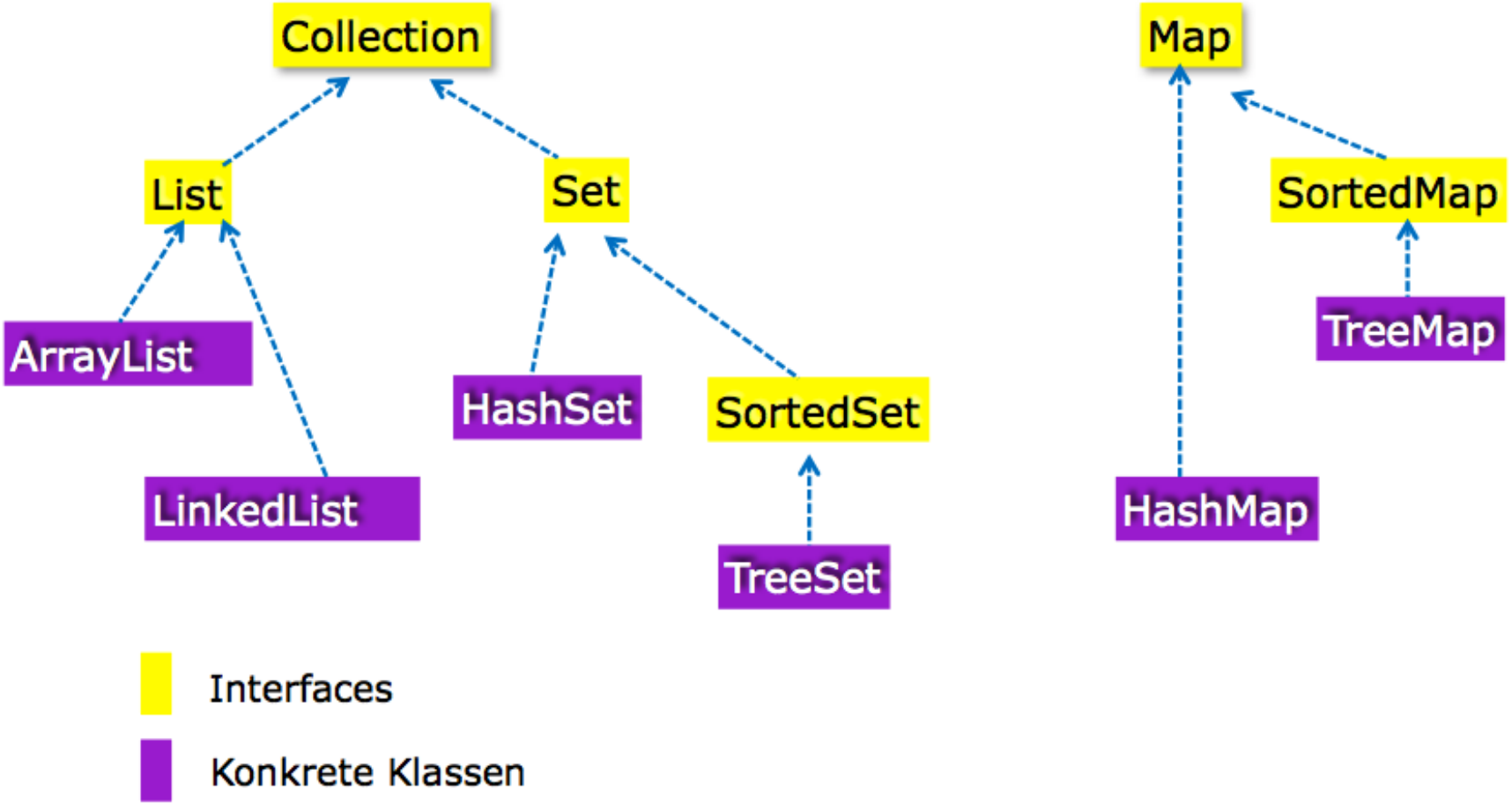


Interfaces als Strukturierungsmittel

- Interfaces in Java können als Unterinterfaces von anderen Interfaces definiert werden.



Collection-Klassen

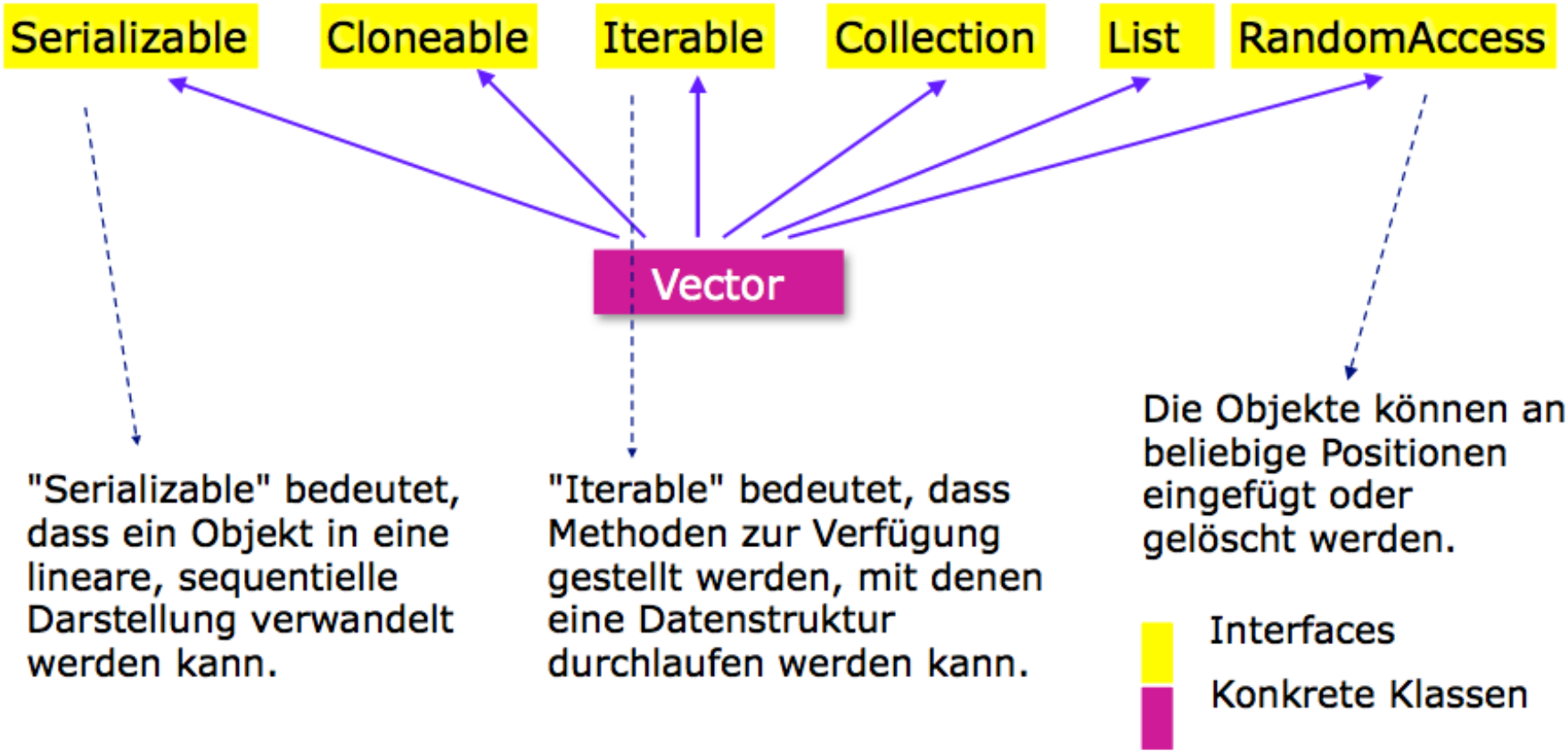


Collections (Sammlungen)

- 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

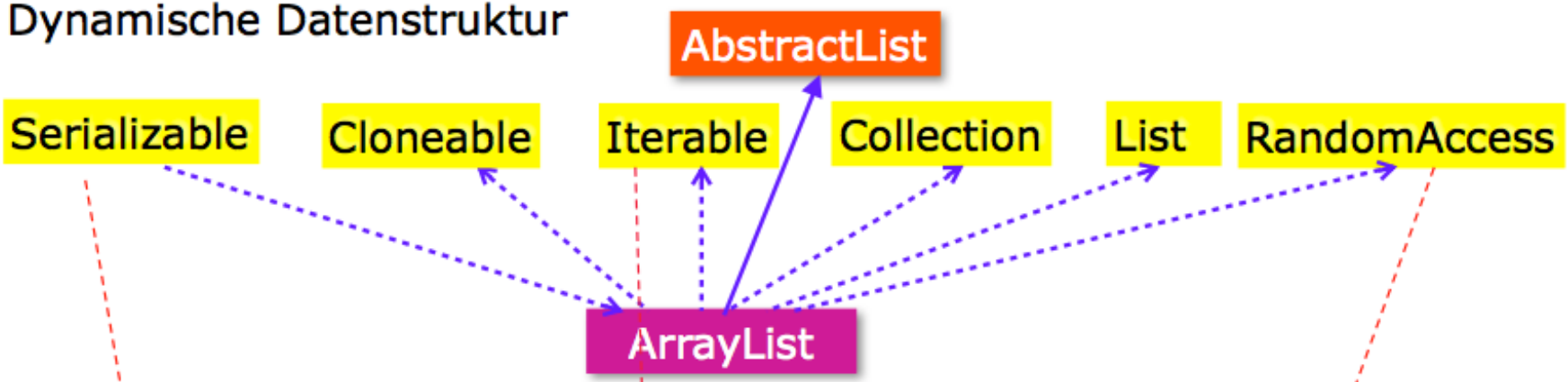
Vektor-Klasse

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

Freie Universität Berlin

Vererbung (inheritance)

ZUSAMMENFASSUNG

Sie sollten wissen...

- warum das Konzept der Vererbung in der Objektorientierung eingeführt wurde.
- was der Unterscheid zwischen Generalisierung und Vererbung ist.
- was mit einfacher bzw. mehrfacher Vererbung gemeint ist.
- was hinter dem Ausdruck „Überladen“ steht.
- warum abstrakte Klassen und Methoden definiert werden.
- was Interfaces in Java sind.