

Objektorientierte Programmierung in Java

Vorlesung 6 - Abstrakte Elemente

Emily Lucia Antosch

HAW Hamburg

24.10.2024

Inhaltsverzeichnis

1. Einleitung	2
2. Abstrakte Elemente & Methoden	6
3. Interfaces	14
4. Vergleich (Interface Comparable)	26
5. License Notice	34

1. Einleitung

- In der letzten Vorlesung ging es um Vererbung
- Sie können nun
 - einfache Vererbungslinien erzeugen und verwenden,
 - Methoden aus der Basisklasse überlagern,
 - die `equals()`-Methode verwenden, um Objekte miteinander zu vergleichen,
 - Objekte über die jeweilige Basisklasse referenzieren
- Heute geht es weiter mit den **Schnittstellen**.

1.1 Wo sind wir gerade?

1. Imperative Konzepte
2. Klassen und Objekte
3. Klassenbibliothek
4. Vererbung
5. **Schnittstellen**
6. Graphische Oberflächen
7. Ausnahmebehandlung
8. Eingaben und Ausgaben
9. Multithreading (Parallel Computing)

- Sie bilden gemeinsame Eigenschaften von Klassen ab, indem Sie Klassen um gemeinsame Schnittstellen (in Form von abstrakten Basisklassen oder Interfaces) erweitern.
- Sie verbergen den Datentyp von Objekten, indem Sie Objekte beim Zugriff auf gemeinsame Eigenschaften unterschiedlicher Klassen über Schnittstellen referenzieren.
- Sie sortieren eine Sammlung von Objekten gleichen Datentyps nach beliebigen Kriterien

2. Abstrakte Elemente & Methoden

? Frage

- Erinnern Sie sich an unsere geometrischen Objekte?
- Was stört Sie am bisherigen Aufbau unserer Klassen?
- Was ergibt keinen Sinn bzw. ist „unschön“?

? Frage

- Erinnern Sie sich an unsere geometrischen Objekte?
- Was stört Sie am bisherigen Aufbau unserer Klassen?
- Was ergibt keinen Sinn bzw. ist „unschön“?

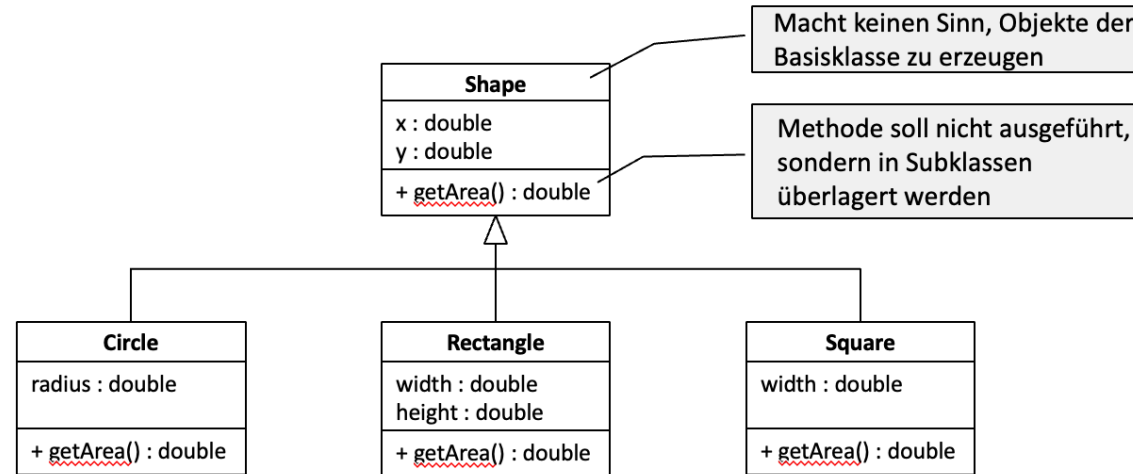



Abbildung 1: Überlagern der Methode aus Shape

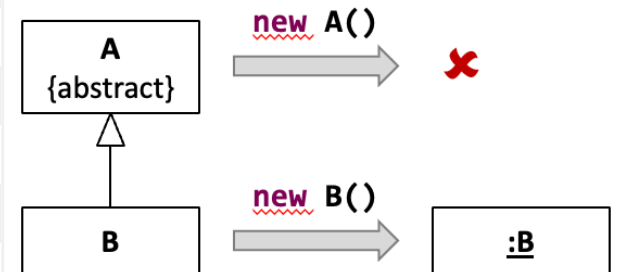
2.1 Abstrakte Elemente

2. Abstrakte Elemente & Methoden

- Klasse wird durch Schlüsselwort `abstract` zu abstrakter Klasse
- Effekt: Es können keine Objekte der Klasse erstellt werden.
- Stattdessen:
 - Klasse ableiten und in (konkreten = nicht abstrakten) Subklassen erweitern
 - Objekte der Subklassen erstellen

```
1  public abstract class A {  
2      // ...  
3  }  
4  
5  public class B extends A {  
6      // ...  
7  }  
8  
9  public static void main(String[] args) {  
10     A abstractObj = new A();  
11     B concreteObj = new B();  
12 }
```

 Java

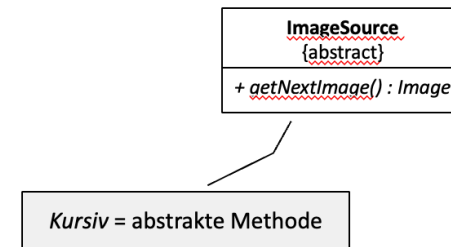
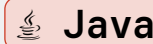


2.1 Abstrakte Elemente

2. Abstrakte Elemente & Methoden

- Methode wird durch das Schlüsselwort `abstract` zur abstrakten Methode
- Abstrakte Methode enthält nur die Deklaration, aber keine Implementierung

```
1 public abstract class ImageSource {  
2     String name;  
3  
4     public abstract Image getNextImage();  
5 }
```



! Merke

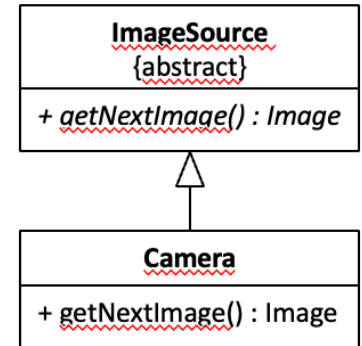
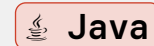
- Abstrakte Methoden können nicht aufgerufen werden (existiert keine Implementierung!)
- Gibt stattdessen vor, welche Methoden Subklassen besitzen müssen

2.1 Abstrakte Elemente

2. Abstrakte Elemente & Methoden

- Klassen mit abstrakten Methoden müssen abstrakt sein.
- Ansonsten könnten für Objekte nicht implementierte Methoden aufgerufen werden.
- Vererbung:
 - Abstrakte Methoden werden vererbt.
 - Subklassen abstrakt, solange nicht alle abstrakten Methoden implementiert

```
1  public abstract class ImageSource {  
2      String name;  
3  
4      public abstract Image getNextImage();  
5  }  
6  
7  public class Camera extends ImageSource {  
8      public Image getNextImage() {  
9          // ...  
10     }  
11 }
```



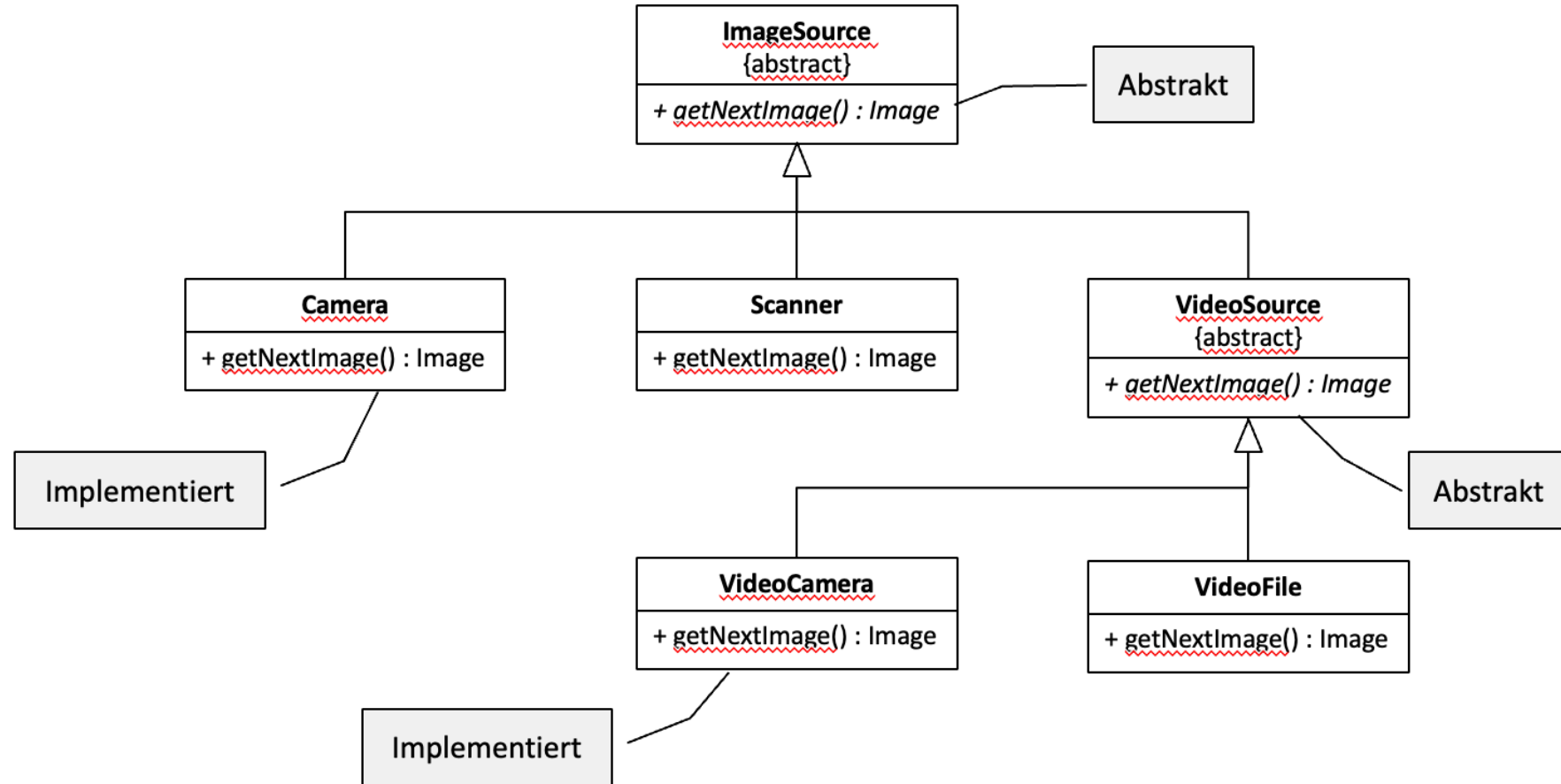
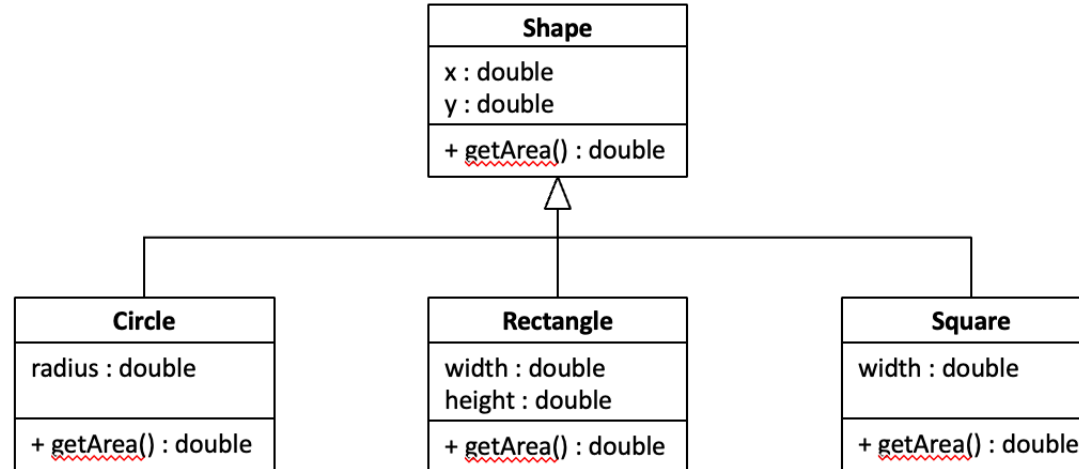


Abbildung 5: Großer Overview über Abstrakte Klassen und Methoden

☰ Aufgabe 1

- Verbessern Sie nun den Aufbau der Klassenstruktur.
- Verwenden Sie hierfür abstrakte Elemente.



2.1 Abstrakte Elemente

2. Abstrakte Elemente & Methoden

- Keine Objekte der Klasse Shape, sondern nur von konkreten geometrischen Formen
- Alle Klassen für geometrischen Formen besitzen `getArea()`.
- Implementierung je nach Typ der geometrischen Form

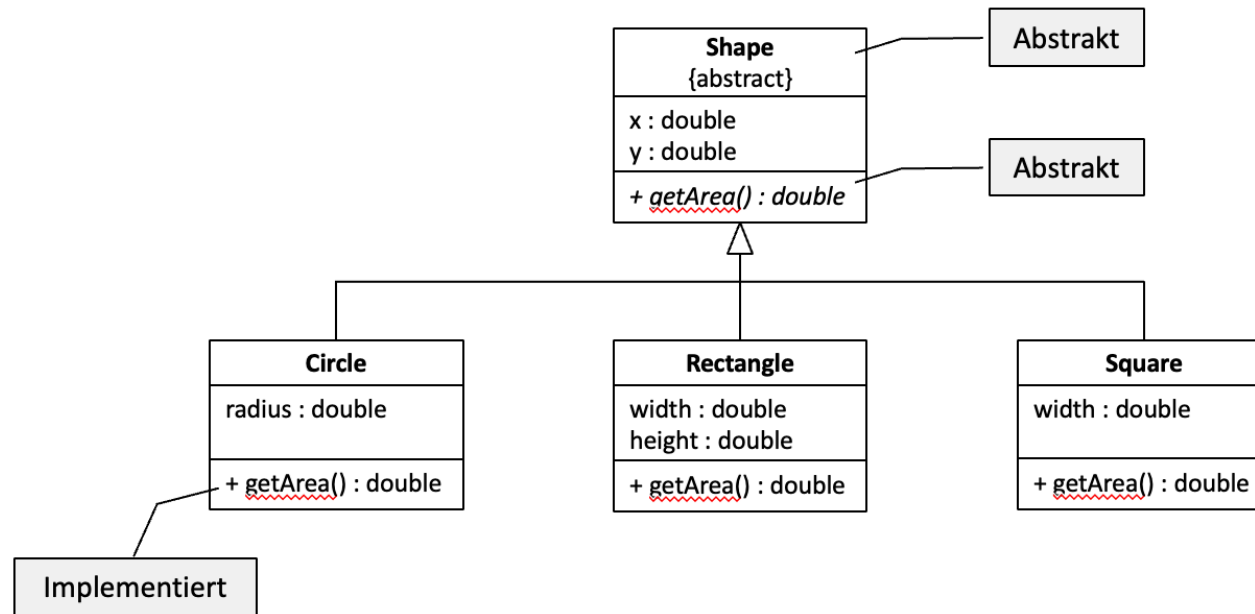


Abbildung 7: Abstrakte und implementierte Methoden

3. Interfaces

3.1 Interfaces

- Klassen (zur Erinnerung):
 - Konkrete Klassen können keine abstrakten Methoden enthalten.
 - Abstrakte Klassen können zusätzlich abstrakte Methoden enthalten.
- Grundlegende Idee einer Schnittstelle:
 - Deklariert lediglich abstrakte Methoden
 - Gibt also vor, welche Methoden eine Klasse implementieren muss
 - Enthält keine Variablen (Kein Objekt erzeugbar: keine Konstruktoren benötigt)
 - Beschreiben oft Eigenschaften (z.B. Comparable, Cloneable, Scalable, ...)

- Sichtbarkeit:
 - Alle Methoden sind public abstract (auch wenn Modifier fehlen).
 - Alle Attribute sind public static final (auch wenn Modifier fehlten).
- Ab Java 8 auch implementierte Methoden:
 - Default-Methoden: Vergleichbar mit herkömmlichen Methoden in einer Klasse
 - Statische Methoden

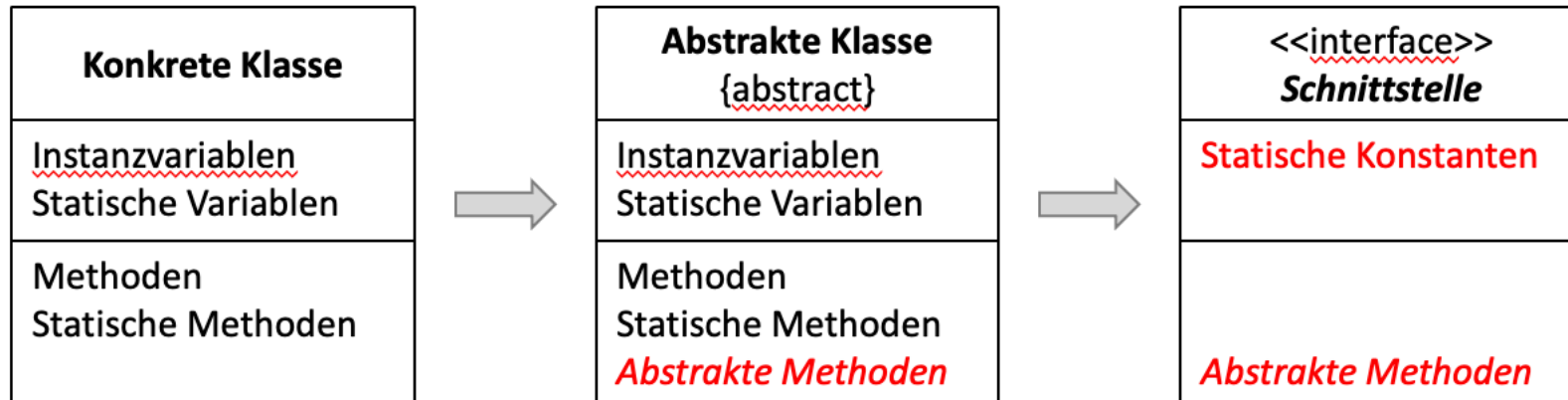


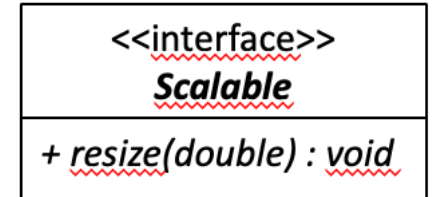
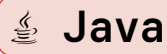
Abbildung 8: Abstufung zwischen Konkret, Abstrakt und Interface

3.1 Interfaces

3. Interfaces

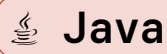
- Deklaration einer Schnittstelle:

```
1  Modifier interface Schnittstellename {  
2      Konstanten  
3      Abstrakte Methoden  
4      Default-Methoden und statische Methoden  
5  }
```



- Deklariere Methode `resize()`, um Größe eines Objektes zu ändern

```
1  public interface Scalable {  
2      void resize(double factor);  
3  }
```

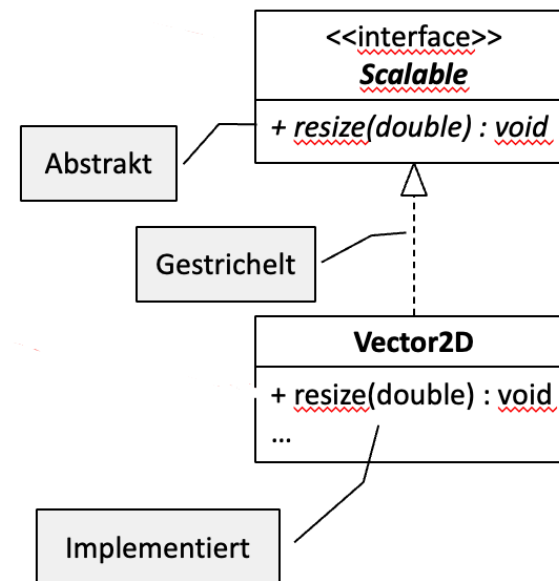


3.1 Interfaces

- Klassen implementieren Schnittstellen über das Schlüsselwort implements
- Klasse erbt Elemente der Schnittstelle und implementiert abstrakte Methoden

```
1  public class Vector2D implements Scalable {
2      private double x, y;
3
4      public Vector2D(double x, double y) {
5          this.x = x;
6          this.y = y;
7      }
8
9      public void resize(double factor) {
10         x *= factor;
11         y *= factor;
12     }
13     // Weitere Methoden ...
14 }
```

Java



3.1 Interfaces

- Interface-Methode nicht implementiert: Methode bleibt abstrakt
- Daher dann auch die Klasse abstrakt
- Subklassen erst dann konkret, wenn alle abstrakten Methoden implementiert

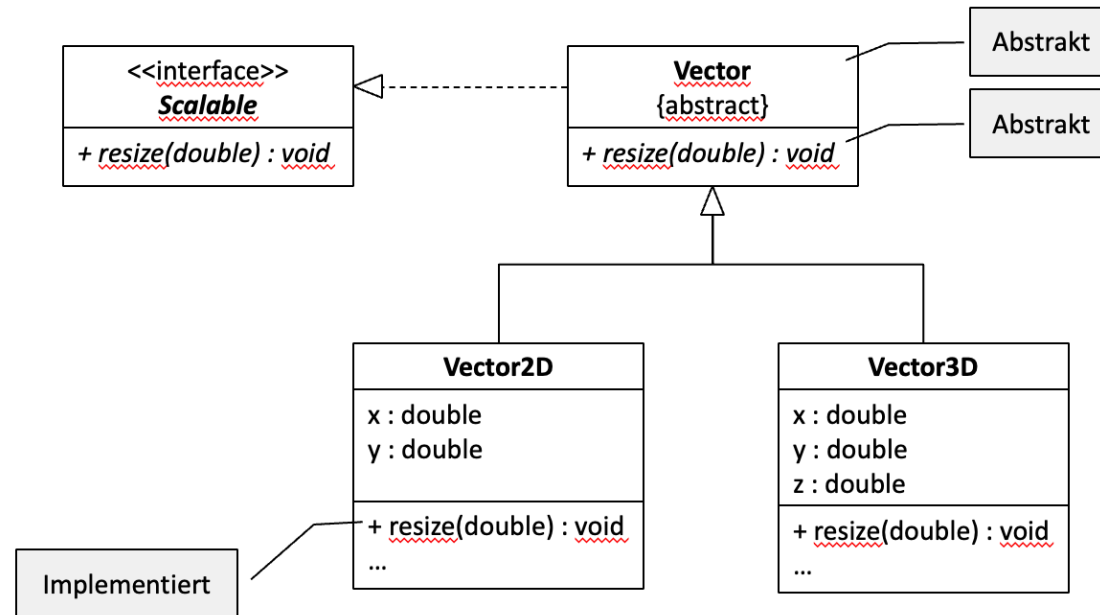


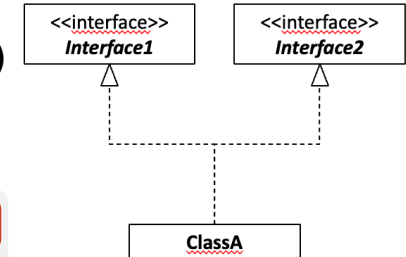
Abbildung 11: Abstrakte Klassen und Interfaces

3.1 Interfaces

3. Interfaces

- Zur Erinnerung: Mehrfachvererbung für Klassen nicht erlaubt
- Aber: Implementierung beliebig vieler Schnittstellen (durch Kommas getrennt) erlaubt

```
1  interface Interface1 {  
2      // ...  
3  }  
4  
5  interface Interface2 {  
6      // ...  
7  }  
8  
9  class ClassA implements Interface1, Interface2 {  
10     // ...  
11 }
```



Klasse GrayImage implementiert Scalable, Drawable und Rotateable

```
1 public class GrayImage implements Scalable, Drawable, Rotateable {  
2     // Attribute und Konstruktoren  
3     // Methoden der Schnittstellen  
4     // Weitere Methoden  
5 }
```

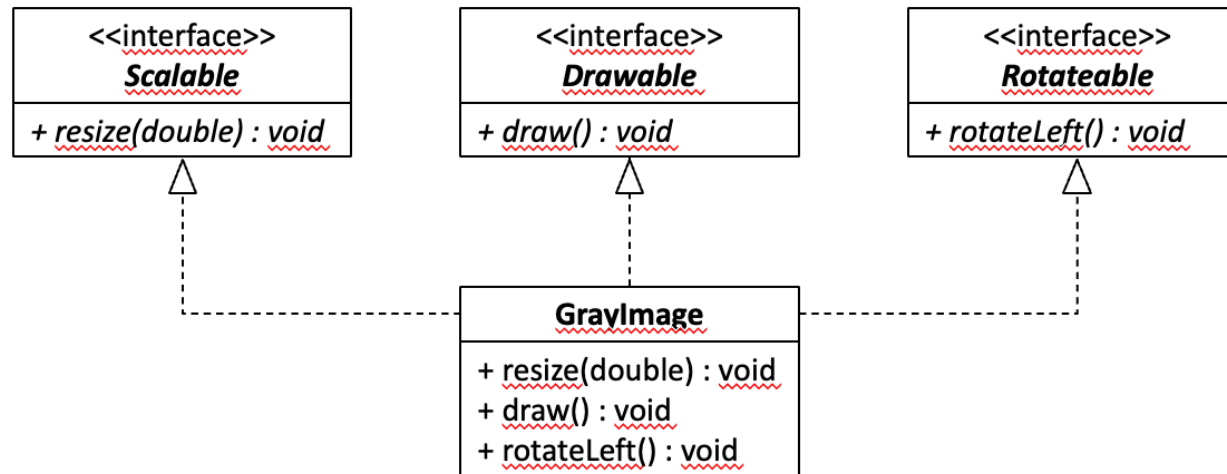
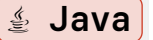
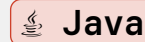


Abbildung 13: GrayImage-Beispiel

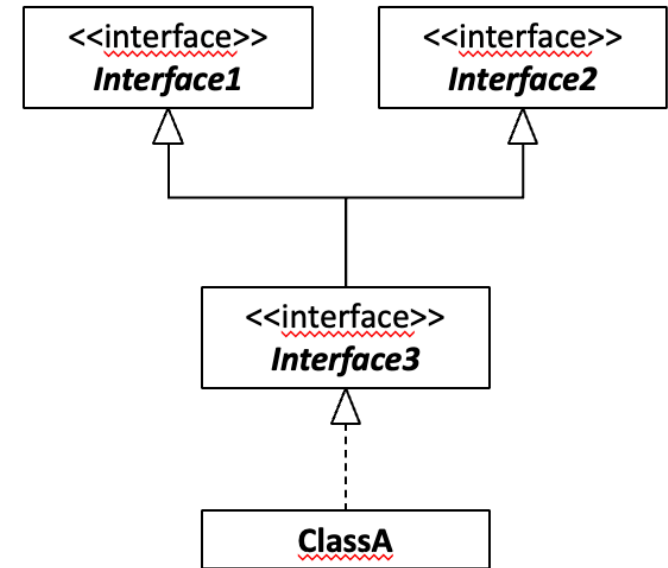
3.1 Interfaces

- Schnittstellen können durch extends abgeleitet werden.
- Für Schnittstellen ist Mehrfachvererbung erlaubt!

```
1  interface Interface1 {  
2      // ...  
3  }  
4  
5  interface Interface2 {  
6      // ...  
7  }  
8  
9  interface Interface3 extends Interface1, Interface2 {  
10     // ...  
11 }  
12  
13 class ClassA implements Interface3 {  
14     // ...  
15 }
```



3. Interfaces




3.1 Interfaces

- Genauso wie bei Basisklassen:
 - Objekte über Datentypen ihrer implementierten Schnittstellen referenzierbar
 - Referenzvariable kann nur auf Attribute und Methoden ihrer Schnittstelle zugreifen

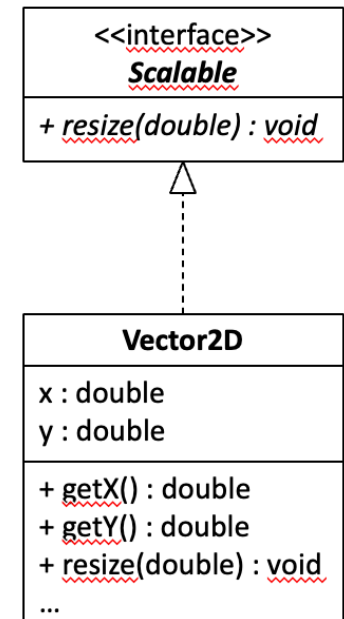
? Frage

- Welche Zugriffe auf Attribute sind zulässig und welche nicht?

```
1 public static void main(String[] args) {
2     Vector2D classRef = new Vector2D(1, 3);
3     Scalable interRef = classRef;
4
5     classRef.resize(1.5);
6     System.out.println(classRef.getX());
7     interRef.resize(1.5);
8     System.out.println(interRef.getX());
9 }
```

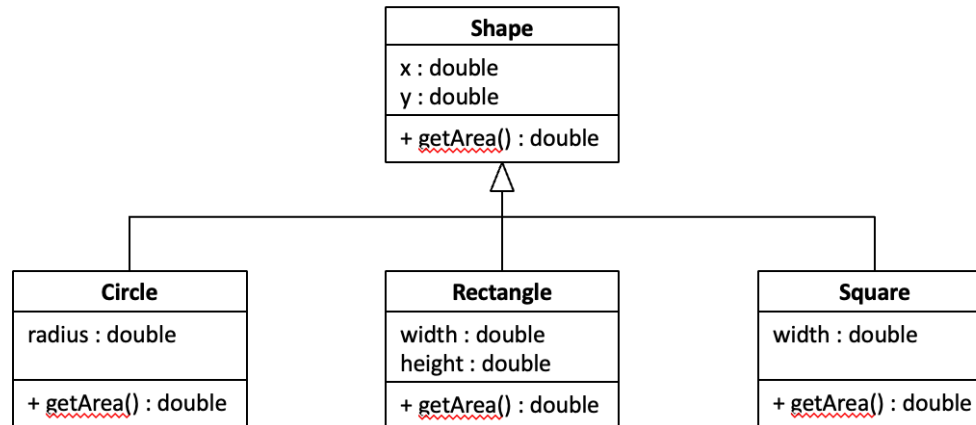
 Java

3. Interfaces



☰ Aufgabe 2

- Erstellen Sie eine Schnittstelle Transformable mit folgenden Methoden:
 - Verschieben
 - Rotation um 90° (je eine Methode für Rotation nach links und nach rechts)
 - Skalieren
 - Implementieren Sie die Schnittstelle in allen Klassen geometrischer Formen



4. Vergleich (Interface Comparable)

4.1 Interface Comparable

4. Vergleich (Interface Comparable)

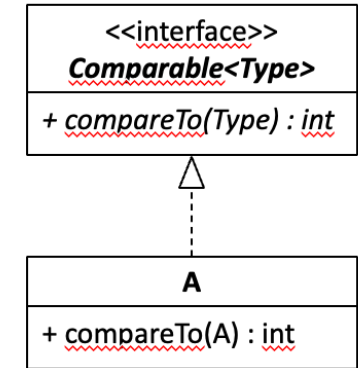
- Vergleich von Objekten (Welches ist „größer“, welches „kleiner“?)

```
1 public interface Comparable<Type> {  
2     public int compareTo(Type other);  
3 }
```



- Verwendung:
 - Schnittstelle in eigener Klasse implementieren
 - Platzhalter Type durch eigenen Klassennamen ersetzen
 - Rückgabewert wird wie folgt gedeutet:

Rückgabewert	Deutung
Negativ	<u>this</u> < <u>other</u>
Null	<u>this</u> == <u>other</u>
Positiv	<u>this</u> > <u>other</u>



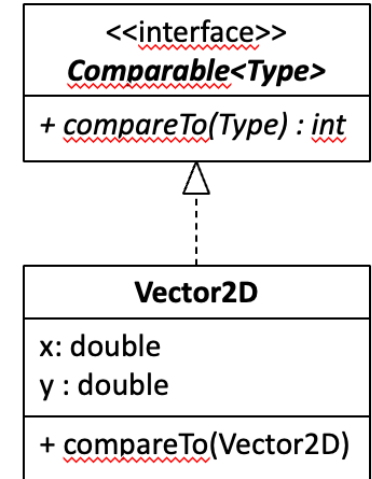
4.1 Interface Comparable

4. Vergleich (Interface Comparable)

- Vergleich von Vektoren anhand des Betrages:

```
1  public class Vector2D implements Comparable<Vector2D> {
2      double x, y;
3
4      public double getAbs() {
5          return Math.sqrt(x * x + y * y);
6      }
7
8      public int compareTo(Vector2D other) {
9          if (getAbs() < other.getAbs()) {
10             return -1;
11          } else if (getAbs() > other.getAbs()) {
12             return 1;
13          } else {
14             return 0;
15          }
16      }
17  }
```

 Java




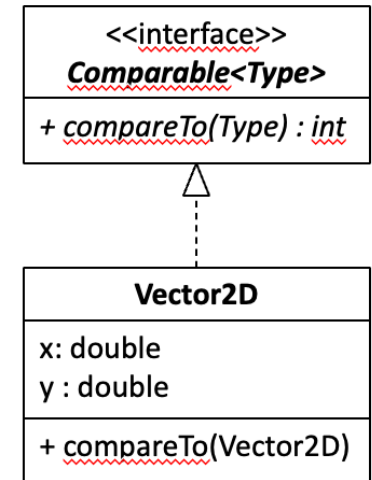
4.1 Interface Comparable

4. Vergleich (Interface Comparable)

- Listen über Klassenmethode `Collections.sort()` sortieren
- Voraussetzung: Elemente in Liste implementieren `Comparable`
- Methode `sort()` verwendet paarweise die Vergleichsmethode `compareTo()`

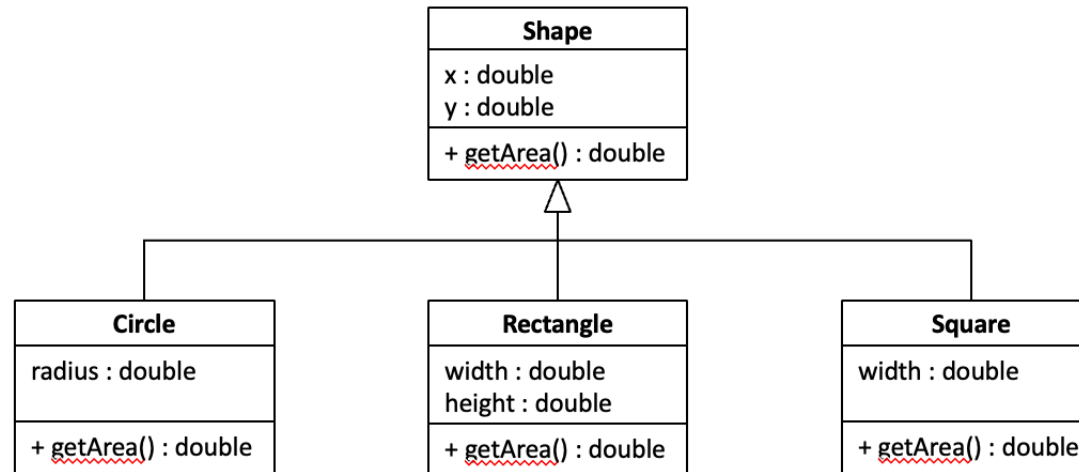
```
1  public static void main(String[] args) {  
2      ArrayList<Vector2D> vectors = new ArrayList<Vector2D>();  
3      vectors.add(new Vector2D(0, 5));  
4      vectors.add(new Vector2D(0, -1));  
5      vectors.add(new Vector2D(7, 8));  
6      vectors.add(new Vector2D(0, 0));  
7  
8      Collections.sort(vectors);  
9      for (Vector2D vector : vectors) {  
10         System.out.println(vector.getAbs());  
11     }  
12 }
```

 Java



☰ Aufgabe 3

- Implementieren Sie Comparable<Type> für geometrische Objekte.
- Kriterium für den Vergleich ist die Fläche der Objekte.

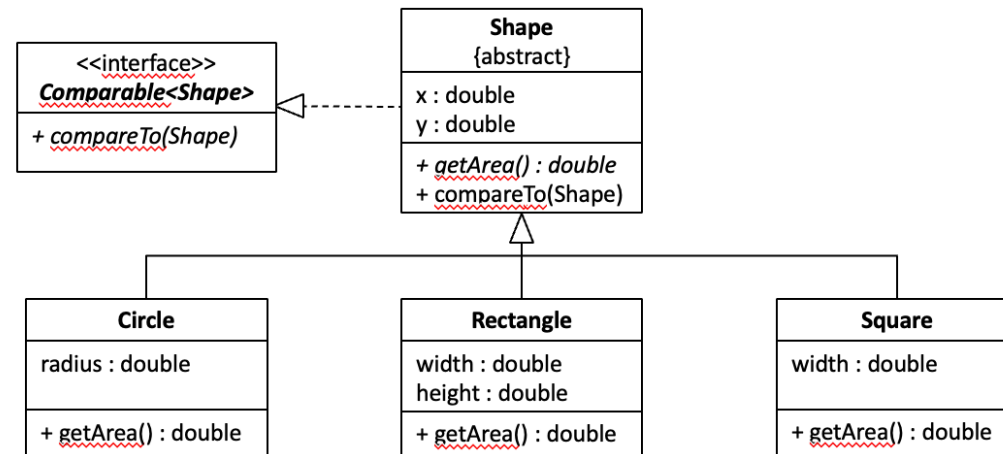


4.1 Interface Comparable

4. Vergleich (Interface Comparable)

! Merke

- Nur die Klasse Shape muss Comparable implementieren.
- Die übrigen Klassen erben die Schnittstelle und Implementierung.

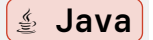


4.1 Interface Comparable

4. Vergleich (Interface Comparable)

- Implementierung in Shape:

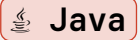
```
1  public abstract class Shape implements Comparable<Shape> {  
2      // Attribute und andere Methoden ...  
3  
4      public int compareTo(Shape other) {  
5          double thisArea = getArea();  
6          double otherArea = other.getArea();  
7  
8          if (thisArea < otherArea) {  
9              return -1;  
10         } else if (thisArea > otherArea) {  
11             return 1;  
12         } else {  
13             return 0;  
14         }  
15     }  
16 }
```



4.1 Interface Comparable

4. Vergleich (Interface Comparable)

```
1  public static void main(String[] args) {
2      ArrayList<Shape> shapes = new ArrayList<Shape>();
3      shapes.add(new Circle(0.0, 0.0, 2.0));
4      shapes.add(new Circle(0.0, 0.0, 1.0));
5      shapes.add(new Rectangle(0.0, 0.0, 10.0, 5.0));
6      shapes.add(new Square(0.0, 0.0, 0.5));
7
8      System.out.println("Flächen (unsortiert):");
9      for (Shape shape : shapes) {
10         System.out.println(shape.getArea());
11     }
12
13     Collections.sort(shapes);
14     System.out.println("\nFlächen (sortiert):");
15     for (Shape shape : shapes) {
16         System.out.println(shape.getArea());
17     }
18 }
```



5. License Notice

5.1 Attribution

- This work is shared under the CC BY-NC-SA 4.0 License and the respective Public License
- <https://creativecommons.org/licenses/by-nc-sa/4.0/>
- This work is based off of the work Prof. Dr. Marc Hensel.
- Some of the images and texts, as well as the layout were changed.
- The base material was supplied in private, therefore the link to the source cannot be shared with the audience.