

Object-Oriented Programming in Java

Lecture 5 - Inheritance

Emily Lucia Antosch

HAW Hamburg

30.06.2025

Contents

1. Introduction	2
2. Inheritance	6
3. Instanziierung von Objekten	39
4. Referenzieren über die Basisklasse	53
5. Überlagern	59
6. Polymorphismus	75
7. Vergleich von Objekten	80
8. License Notice	89

1. Introduction

1.1 Where Are We Currently?

- In der letzten Vorlesung ging es um Klassenbibliotheken
- Sie können nun
 - einfache Klassenbibliotheken verwenden, um Strings oder Arrays zu manipulieren
 - aus einem Array alle Items über eine foreach-Schleife auslesen
 - mithilfe von Wrapperklassen Typumwandlungen durchführen,
 - einfache mathematische Rechnung mithilfe der Math-Klasse ausführen.
- Heute geht es weiter mit der **Vererbung**.

1.1 Where Are We Currently?

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 erzeugen neue Datentypen, indem Sie bestehende Klassen um zusätzliche Eigenschaften erweitern, um beispielsweise duplizierten Quelltext zu vermeiden.
- Sie verwenden Sichtbarkeits-Modifizierer, um die Attribute einer Klasse vor direktem Zugriff von außen zu schützen.

2. Inheritance

2.1 Inheritance

- Klasse übernimmt („erbt“) Variablen und Methoden einer vorhandener Klassen
- Ziel: Wiederverwendung existierender Klassen
- Beispiel und UML-Notation:
 - Klasse A ist vorhanden
 - Klasse B wird erstellt und erbt von A
- Begriffe:
 - Klasse A: Superklasse (Basisklasse, Oberklasse)
 - Klasse B: Subklasse (abgeleitete Klasse, Unterklasse)
 - Vererbung: Ableitung, engl.: inheritance

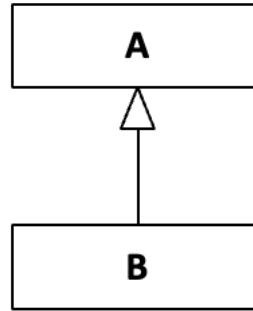


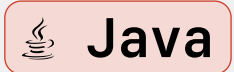
Figure 1: Einfaches Beispiel für Vererbung

2.1 Inheritance

2. Inheritance

- Ableitung der Basisklasse mittels extends:

```
1  class Klassenname extends Basisklasse {  
2      Attribute  
3      Methoden  
4  }
```





Example

```
1  class A {  
2      // ...  
3  }  
4  
5  class B extends A {  
6      // ...  
7  }
```



Task 1

- Erstellen Sie folgende Klassen:
 - Person: Objekte beinhalten den Namen
 - Pilot: Objekte beinhalten den Namen und die bisherigen Flugstunden
 - Ausführbare Klasse, die ein Objekt Pilot erzeugt und den Namen ausgibt

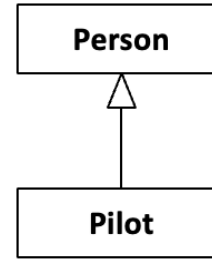
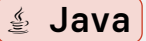


Figure 2: Pilot erbt von Person

2.1 Inheritance

2. Inheritance

```
1  public class Person {
2      String name;
3  }
4
5  public class Pilot extends Person {
6      int flightHours;
7  }
8
9  public class PilotDemo {
10     public static void main(String[] args) {
11         Pilot pilot = new Pilot();
12
13         pilot.name = "Lukas Luft";
14         pilot.flightHours = 1482;
15         System.out.println("Name: " + pilot.name);
16     }
17 }
```



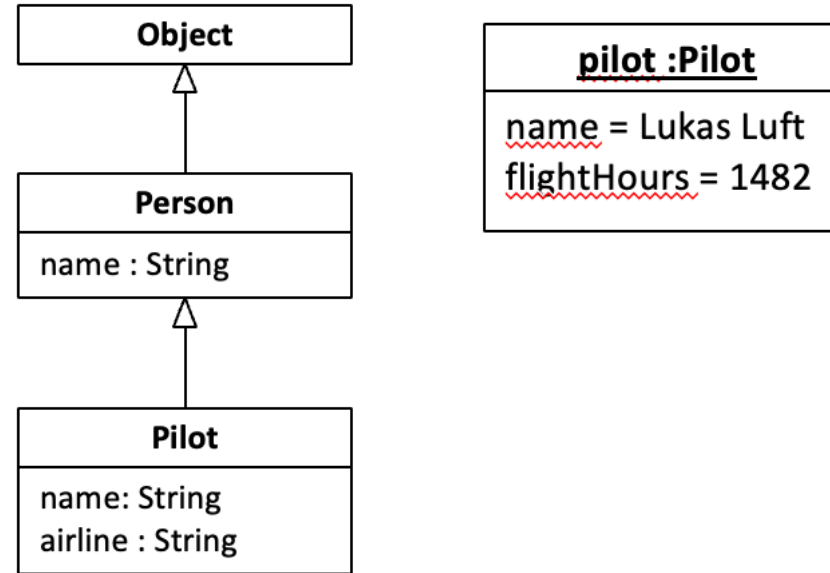
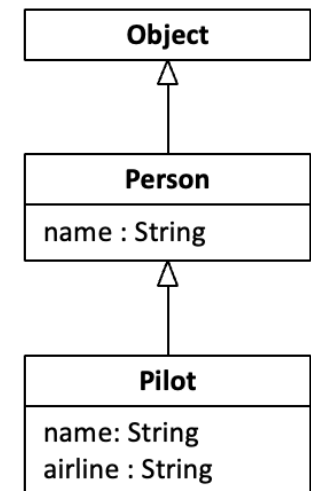
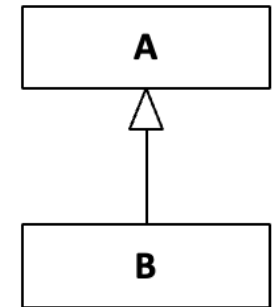


Figure 3: Vererbung von Attributen

2.1 Inheritance

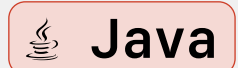
- Klasse B kann neue Variablen und Methoden hinzufügen
- Begriffe:
 - Spezialisierung: Klasse B ist spezieller als Klasse A
 - Generalisierung: Klasse A ist allgemeiner als Klasse B
- Beispiel:
 - Klasse Pilot hat von Person geerbt und flightHours hinzugefügt



2.1 Inheritance

- Ein Pilot ist eine Person, d.h. Person ist allgemeiner als Pilot.

```
1  public class Person {  
2      String name;  
3  }  
4  
5  public class Pilot extends Person {  
6      int flightHours;  
7  }
```



2.1 Inheritance

- Datenkapselung (information hiding): Variablen vor Zugriff von außen geschützt
- Einschränkungen des Zugriffs auf Klassen, Variablen und Methoden durch Modifizierer
- Gedankenbild: „Sichtbarkeit“ (d.h. ist Element sichtbar bzw. bekannt?)

<u>Modifizierer</u>	UML	Wo sichtbar?	Klassen	Variablen	Methoden
<u>public</u>	+	Alle Klassen	x	x	x
<u>protected</u>	#	Klassen des eigenen Paketes, Unterklassen		x	x
private	-	Nur innerhalb der eigenen Klasse		x	x
<keiner> *	~	Klassen des eigenen Paketes	x	x	x

* „Default“-Modifizierer

Figure 5: Modifier für Klassen, Methoden und Attribute

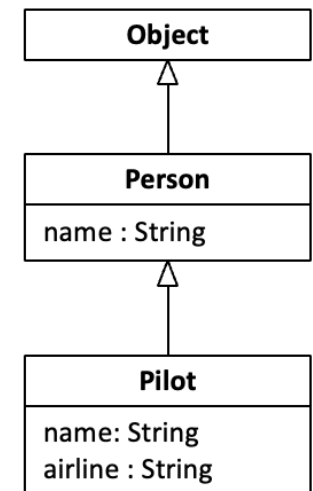
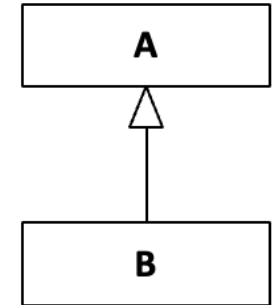


Question

Ist `private` als Modifier für Konstruktoren erlaubt?

2.1 Inheritance

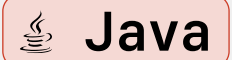
- Klasse B erbt alle sichtbaren Variablen und Methoden der Klasse A
- Klasse B besitzt Variablen und Methoden von A und kann diese verwenden (so als ob diese in Klasse B definiert worden wären)
- Beispiel:
 - Objekt von Klasse Pilot nutzt Variable name der Basisklasse Person



2.1 Inheritance

2. Inheritance

```
1  public class PilotDemo {
2      public static void main(String[] args) {
3          Pilot pilot = new Pilot();
4
5          pilot.name = "Lukas Luft";
6          pilot.flightHours = 1482;
7          System.out.println("Name: " + pilot.name);
8      }
9  }
```



? Question

- Was meinen Sie, welche Bestandteile einer Klasse werden nicht vererbt?
- Nicht an abgeleitete Klasse weitergegeben:
 - Konstruktoren und Destruktoren
 - Klassenvariablen und Klassenmethoden (Modifizierer static)
 - Private Variablen und Methoden (Modifizierer private)

- Hinweise:
 - Statische Elemente nie vererbt, da an eine Klasse und nicht an konkretes Objekt gebunden
 - Private Elemente sind in Subklasse vorhanden, sie kann aber nicht direkt darauf zugreifen

2.1 Inheritance

- Subklassen können weitervererbt werden.
- Von einer Klasse können beliebig viele Subklassen abgeleitet werden.
- Das Erben von mehreren Basisklassen ist hingegen nicht möglich (Mehrfachvererbung)

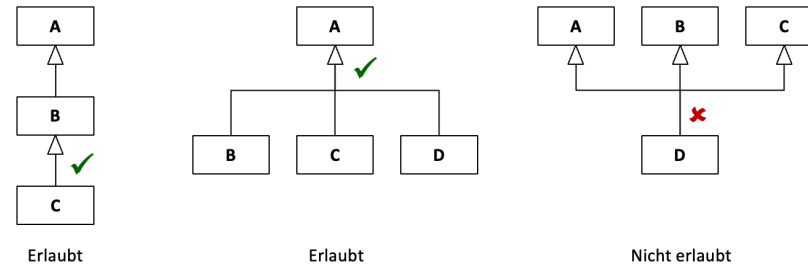
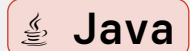


Figure 7: Mögliche Strukturen zur Vererbung

? Question

- Was meinen Sie, welche Basisklasse besitzt Person?

```
1  public class Person {  
2      String name;  
3  }
```



Java

- Sie konnten es bisher nicht wissen:
 - In Java ist eine Klasse Object definiert.
 - Keine Basisklasse angegeben. Implizit von Object abgeleitet (extends Object)

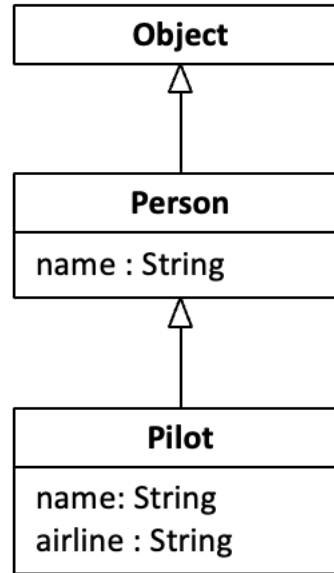


Figure 8: Object als Superklasse von Person

2.1 Inheritance

- Wichtige Konsequenz:
 - Object ist Basisklasse jeder Vererbungshierarchie

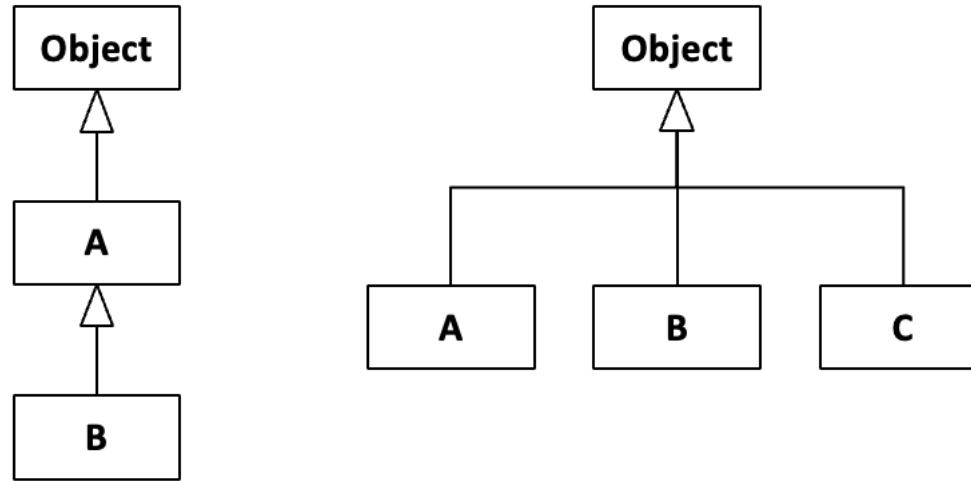


Figure 9: Object als Teil jeder Vererbung

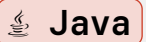
? Question

- Was meinen Sie?
 - Wie viele Klassen besitzen keine Basisklasse?
 - Wie viele Klassen besitzen mehr als eine direkte Basisklasse?

2.1 Inheritance

- Wichtige Konsequenz:
- Jede Klasse erbt die in Object definierten Methoden (z. B. toString())
- Beispiel:

```
1  public class Person {  
2      String name;  
3  }  
4  
5  public class ObjectDemo {  
6      public static void main(String[] args) {  
7          Person person = new Person();  
8  
9          person.name = "Lukas Luft";  
10         System.out.println(person.toString());  
11     }
```



```
12 }
```

Task 2

- Implementieren Sie Klassen für geometrische Objekte Kreis, Rechteck und Quadrat.
- Verwenden Sie zunächst nur öffentliche Variablen.
- Implementieren Sie zunächst keine Methoden.

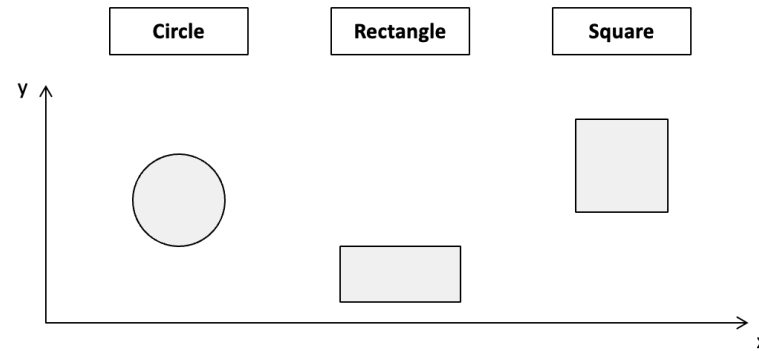


Figure 10: Geometrische Formen als Objekte

```
1  public class Circle {
2      public double x, y;
3      public double radius;
4  }
5
6  public class Rectangle {
7      public double x, y;
8      public double width, height;
9  }
10
11 public class Square {
12     public double x, y;
13     public double width;
14 }
```

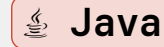


☰ Task 3

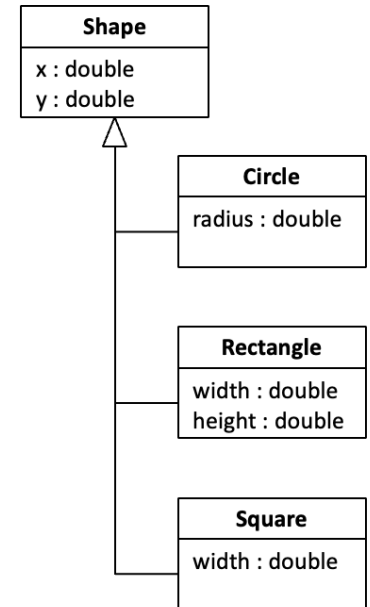
Erstellen Sie nun eine gemeinsame Basisklasse!

2.1 Inheritance

```
1  public class Shape {
2      public double x, y;
3  }
4
5  public class Circle extends Shape {
6      public double radius;
7  }
8
9  public class Rectangle extends Shape {
10     public double width, height;
11 }
12
13 public class Square extends Shape {
14     public double width;
15 }
```



2. Inheritance



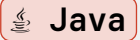
? Question

- Welche Variablen sind in den jeweiligen Klassen deklariert?

☰ Task 4

- Ergänzen Sie für die Klasse Circle einen Konstruktor!

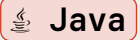
```
1  public class Shape {
2      public double x, y;
3  }
4
5  public class Circle extends Shape {
6      public double radius;
7
8      public Circle(double x, double y, double radius) {
9          this.x = x;
10         this.y = y;
11         this.radius = radius;
12     }
13 }
```



! Memorize

- Beachte: Variablen x und y der Basisklasse werden wie „eigene“ Variablen verwendet
- Verstecken Sie die Variablen der Klasse Shape durch den Modifizierer private.

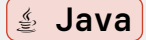
```
1  public class Shape {
2      private double x, y;
3  }
4
5  public class Circle extends Shape {
6      public double radius;
7
8      public Circle(double x, double y, double radius) {
9          this.x = x;
10         this.y = y;
11         this.radius = radius;
12     }
13 }
```



✗ Error

- Die Variablen x und y der Basisklasse sind in Circle nicht sichtbar.
- Fehler: Im Konstruktor der Klasse Circle sind x und y unbekannt.

```
1  public class Shape {
2      private double x, y;
3
4      public void setX(double x) {
5          this.x = x;
6      }
7      // Zusätzlich Getter sowie entsprechende Methoden für y ...
8  }
9
10 public class Circle extends Shape {
11     public double radius;
12
13     public Circle(double x, double y, double radius) {
14         setX(x);
15         setY(y);
16         this.radius = radius;
17     }
18 }
```



3. Instanziierung von Objekten

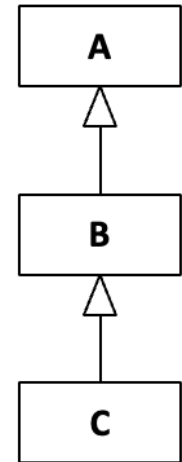
3.1 Erzeugung von Objekten

3. Instanziierung von Objekten

- Klasse C enthält eigene Methoden sowie Methoden der Klassen A und B.
- Klasse C enthält eigene Variablen sowie Variablen der Klassen A und B.

? Question

- Was meinen Sie?
 - Wie werden Methoden eines neuen Objektes der Klasse C erzeugt?
 - Wie werden Variablen eines Objektes der Klasse C erzeugt und initialisiert?



- Methoden:
 - Werden nicht für jedes Objekt neu erzeugt, sondern sind für Klasse definiert
- Variablen:
 - An Basisklasse der Vererbungshierarchie beginnen
 - In jedem Schritt Variablen der entsprechenden (Basis-)Klasse erzeugen und initialisieren

3.1 Erzeugung von Objekten

3. Instanziierung von Objekten

- Initialisierung über Konstruktor der jeweiligen (Basis-)Klasse

3.1 Erzeugung von Objekten

3. Instanziierung von Objekten

- Variablen für Objekte der Klasse C:
 - Objekt enthält die in der Klasse C deklarierten Variablen
 - Enthält zusätzlich von Klasse B geerbte Variablen
 - Diese enthalten die von Klasse A geerbten Variablen

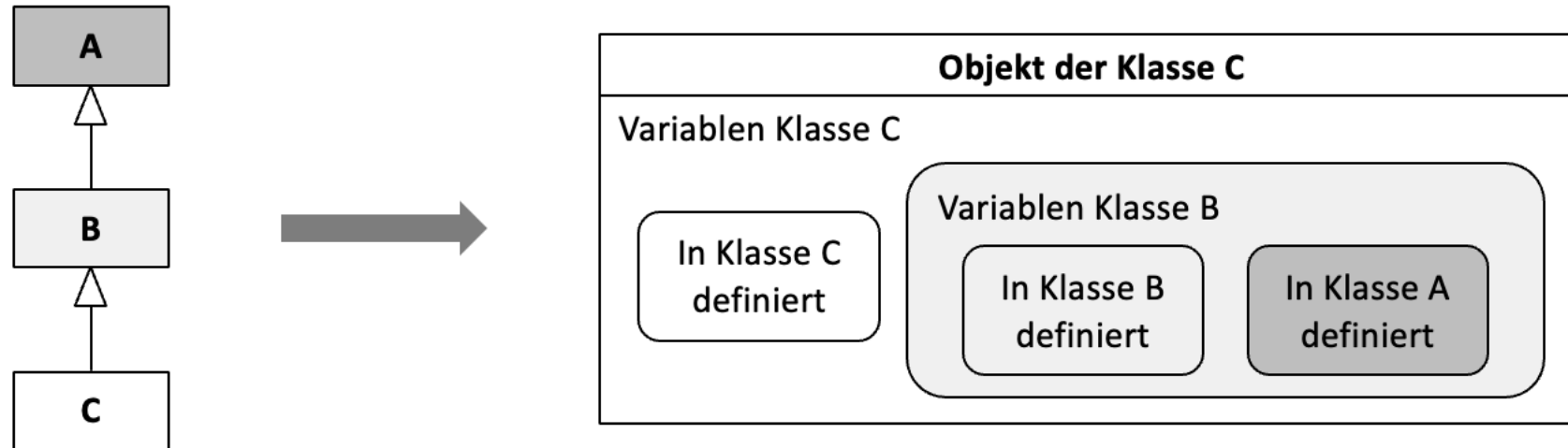


Figure 13: Zusammensetzung des Objekts der Klasse C

- Neues Objekt der Klasse C:
 - Vererbungshierarchie nach oben durchlaufen:
 - Klasse C hat Basisklasse B: Aufruf, um Variablen von B zu erzeugen
 - Klasse B hat Basisklasse A: Aufruf, um Variablen von A zu erzeugen
- Variablen „von innen nach außen“ erzeugen und initialisieren (Konstruktorverkettung):
 - Variablen von A erzeugen und über Konstruktor A() initialisieren

- Variablen von B erzeugen und über Konstruktor B() initialisieren
- Variablen von C erzeugen und über Konstruktor C() initialisieren

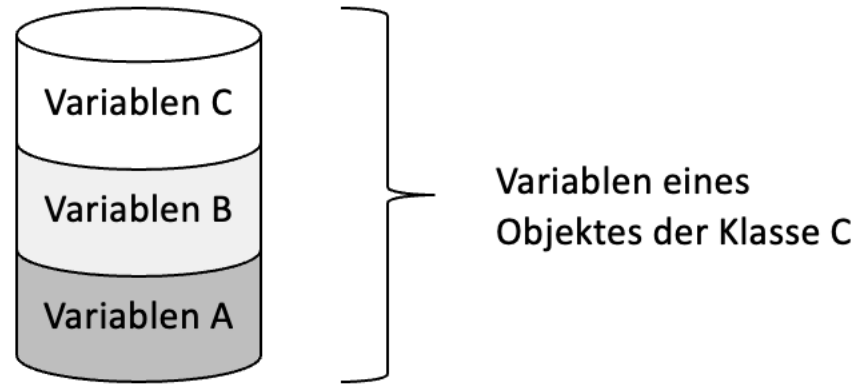
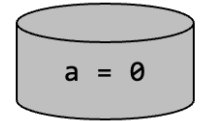


Figure 14: Variablen der Klasse C im Speicher

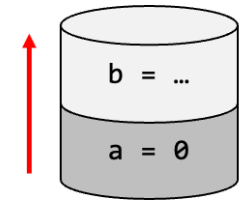
3.1 Erzeugung von Objekten

3. Instanziierung von Objekten

- Konstruktor der Basisklasse:
 - Aufruf über `super()` als erste Anweisung im Konstruktor der abgeleiteten Klasse
 - Fehlt `super(...)` wird der Standardkonstruktor der Basisklasse aufgerufen.



```
1  public class A {  
2      double a;  
3      // Standardkonstruktor wird automatisch erzeugt  
4  }  
5  
6  public class B extends A {  
7      double b;  
8  
9      public B(double b) {  
10         super();    // Aufruf Standardkonstruktor Klasse A  
11         this.b = b;  
12     }  
13 }
```

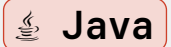


3.1 Erzeugung von Objekten

3. Instanziierung von Objekten

- Sie erinnern sich?

```
1  public class Shape {
2      private double x, y;
3  }
4
5  public class Circle extends Shape {
6      public double radius;
7
8      public Circle(double x, double y, double radius) {
9          this.x = x;
10         this.y = y;
11         this.radius = radius;
12     }
13 }
```



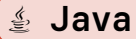
Task 5

- Die Variablen x und y sind in Circle unbekannt.
- Lösen Sie das Problem durch Ergänzen eines Konstruktors für die Basisklasse Shape.

3.1 Erzeugung von Objekten

3. Instanziierung von Objekten

```
1  public class Shape {
2      private double x, y;
3
4      public Shape(double x, double y) {
5          this.x = x;
6          this.y = y;
7      }
8  }
9
10 public class Circle extends Shape {
11     public double radius;
12
13     public Circle(double x, double y, double radius) {
14         super(x, y); // Passende Signatur zum Konstruktor der Basisklasse!
15         this.radius = radius;
16     }
17 }
```



☰ Task 6

- Schützen Sie alle Attribute durch den Modifizierer `private`.
- Erzeugen Sie gegebenenfalls geeignete Getter und Setter.

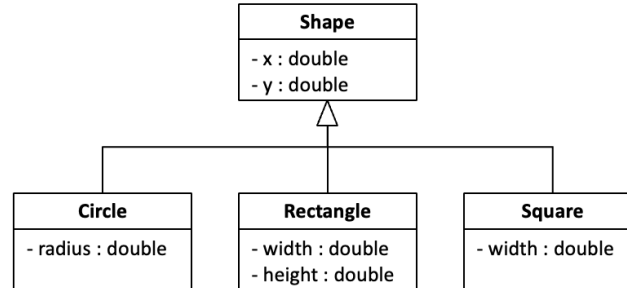
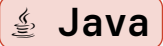


Figure 16: Aufbau der Vererbung

3.1 Erzeugung von Objekten

3. Instanziierung von Objekten

```
1  public class Circle extends Shape {
2      private double radius;
3
4      public Circle(double x, double y, double radius) {
5          super(x, y);
6          this.radius = radius;
7      }
8
9      public double getRadius() {
10         return radius;
11     }
12
13     public void setRadius(double radius) {
14         this.radius = radius;
15     }
```



Java

3.1 Erzeugung von Objekten

3. Instanziierung von Objekten

```
16 }
```

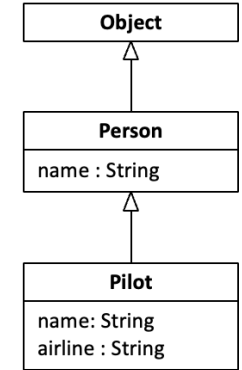
4. Referenzieren über die Basisklasse

4.1 Referenz Basisklasse

4. Referenzieren über die Basisklasse

- Betrachten wir folgende Vererbungslinie:

```
1  public class Person {
2      String name;
3
4      public Person(String name) {
5          this.name = name;
6      }
7  }
8
9  public class Pilot extends Person {
10     String airline;
11
12     public Pilot(String name, String airline) {
13         super(name);
14         this.airline = airline;
15     }
16 }
```

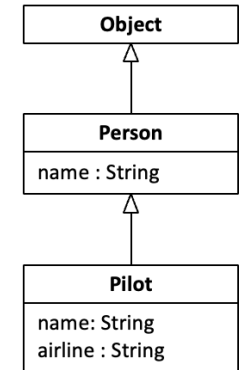


4.1 Referenz Basisklasse

4. Referenzieren über die Basisklasse

- Was meinen Sie zu folgendem Programm?

```
1  public class ReferenceDemo {
2      public static void main(String[] args) {
3          Pilot pilot = new Pilot("Birgit", "Winglet Airways");
4          Person personRef = pilot;
5          Object objectRef = pilot;
6
7          System.out.println(personRef.name);
8      }
9  }
```



! Memorize

Es gibt nur ein Objekt (mit Datentyp Pilot). Objekt wird über Variablen mit anderen Datentypen als Pilot referenziert

4.1 Referenz Basisklasse

4. Referenzieren über die Basisklasse

- Klasse Pilot erbt von Klasse Person und erweitert diese
- Pilot beinhaltet Person („Pilot is a Person“) Als Person referenzierbar
- Objekt wird hierdurch nicht verändert (d.h. Objekt bleibt vom Typ Pilot)!

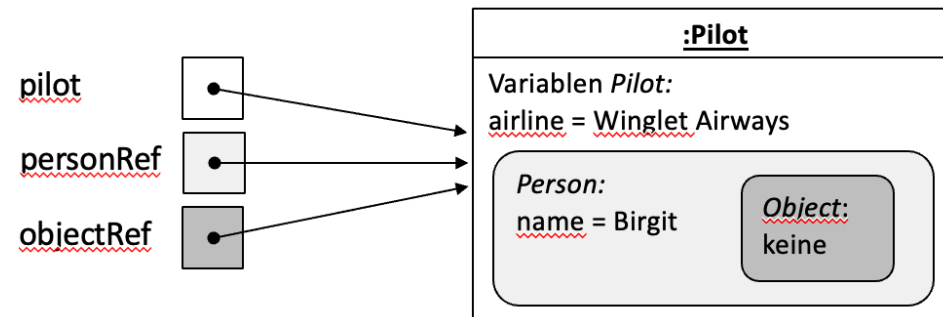


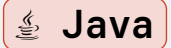
Figure 19: Referenzen auf ein Objekt mit Basisklasse

- Allgemein:
 - Objekte können wie Objekte ihrer Basisklassen behandelt werden.
 - Objekte über Datentypen ihrer Basisklassen referenzierbar
 - Referenzvariable kann nur auf Attribute und Methoden ihrer Klasse zugreifen

? Question

- Welche Zugriffe auf Attribute sind zulässig und welche nicht?

```
1  public static void main(String[] args) {  
2      Pilot pilot = new Pilot("Birgit", "Winglet Airways");  
3      Person personRef = pilot;  
4      Object objectRef = pilot;  
5  
6      System.out.println(personRef.name);  
7      System.out.println(personRef.airline);  
8      System.out.println(objectRef.name);  
9      System.out.println(objectRef.airline);  
10 }
```



5. Überlagern

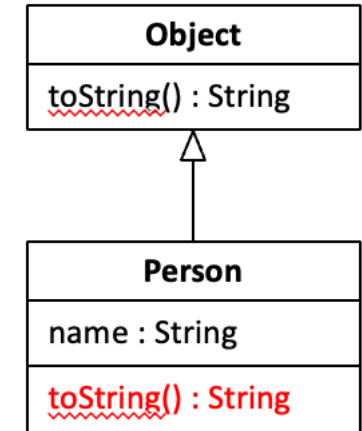
5.1 Überlagern von Methoden

- Klasse Person ist von Object abgeleitet und erbt daher toString() von Object und definiert daher eine weitere toString()-Methode

```
1  public class Person {
2      private String name;
3
4      public Person(String name) {
5          this.name = name;
6      }
7
8      public String getName() {
9          return name;
10     }
11
12     public String toString() {
13         return name;
14     }
15 }
```



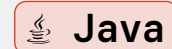
5. Überlagern



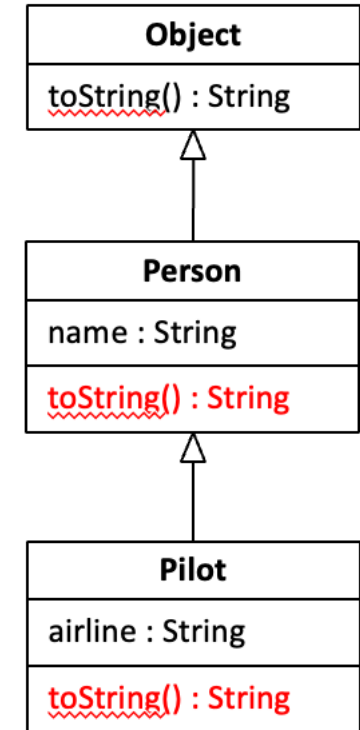
5.1 Überlagern von Methoden

- Klasse Pilot:
 - Von Person abgeleitet und erbt daher toString() von Person
 - Definiert noch eine toString()-Methode

```
1  public class Pilot extends Person {
2      private String airline;
3
4      public Pilot(String name, String airline) {
5          super(name);
6          this.airline = airline;
7      }
8
9      public String toString() {
10         return String.format("%s (%s)", getName(), airline);
11     }
12 }
```



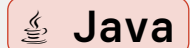
5. Überlagern



? Question

Was wird ausgegeben?

```
1 public static void main(String[] args) {  
2     Person person = new Person("Birgit Janssen");  
3     System.out.println("person: " + person);  
4     System.out.println("person.toString(): " + person.toString());  
5  
6     Pilot pilot = new Pilot("Jan Birgerson", "Winglet Airways");  
7     System.out.println("pilot.toString(): " + pilot.toString());  
8 }
```

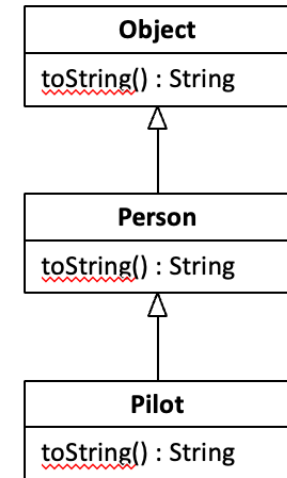


5.1 Überlagern von Methoden

- Ausgabe:
 - person: Birgit Janssen
 - person.toString(): Birgit Janssen
 - pilot.toString(): Jan Birgerson (Winglet Airways)

! Memorize

- Jeweils Methode der entsprechenden Klasse, nicht der Superklasse(n), ausgeführt
- Begriff: Methode der Superklasse durch neu definierte Methoden überlagert



? Question

- Was wird ausgegeben?

```
1 public static void main(String[] args) {  
2     Pilot pilot = new Pilot("Jan Birgerson", "Winglet Airways");  
3     Object objectRef = pilot;  
4     Person personRef = pilot;  
5  
6     System.out.println("objectRef: " + objectRef.toString());  
7     System.out.println("personRef: " + personRef.toString());  
8 }
```



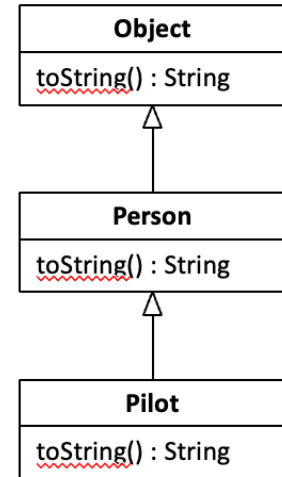
5.1 Überlagern von Methoden

5. Überlagern

? Question

- Was wird ausgegeben?

```
1 public static void main(String[] args) {  
2     Pilot pilot = new Pilot("Jan Birgerson", "Winglet Airways");  
3     Object objectRef = pilot;  
4     Person personRef = pilot;  
5  
6     System.out.println("objectRef: " + objectRef.toString());  
7     System.out.println("personRef: " + personRef.toString());  
8 }
```



- Ausgabe:
 - objectRef: Jan Birgerson (Winglet Airways)
 - personRef: Jan Birgerson (Winglet Airways)

! Memorize

Methode der entsprechenden Klasse ausgeführt, selbst bei Referenz über Superklasse(n)

5.1 Überlagern von Methoden

5. Überlagern

- Zugriff auf überlagerte Methoden der Basisklasse über Referenz super
- Beispiel:

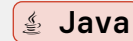
```
1  public class Pilot extends Person {
2      // Instanzvariable, Konstruktor, toString() ...
3
4      public String toStringOfSuperClass() {
5          return super.toString();
6      }
7  }
8
9  public static void main(String[] args) {
10     Pilot pilot = new Pilot("Jan Birgerson", "Winglet Airways");
11     System.out.println("Pilot.toString(): " + pilot.toString());
12     System.out.println("super.toString(): " + pilot.toStringOfSuperClass());
13 }
```



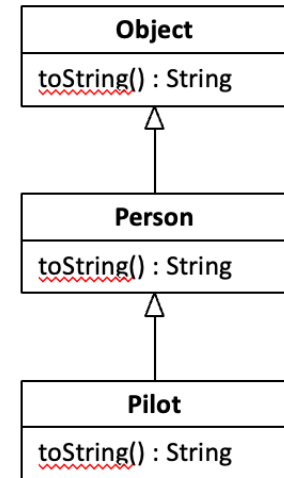
5.1 Überlagern von Methoden

- Zugriff auf überlagerte Methoden der Basisklasse über Referenz super
- Beispiel:

```
1  public class Pilot extends Person {
2      // Instanzvariable, Konstruktor, toString() ...
3
4      public String toStringOfSuperClass() {
5          return super.toString();
6      }
7  }
8
9  public static void main(String[] args) {
10     Pilot pilot = new Pilot("Jan Birgerson", "Winglet Airways");
11     System.out.println("Pilot.toString(): " + pilot.toString());
12     System.out.println("super.toString(): " + pilot.toStringOfSuperClass());
13 }
```



5. Überlagern



- Ausgabe:
 - `Pilot.toString(): Jan Birgerson (Winglet Airways)`
 - `super.toString(): Jan Birgerson`

5.1 Überlagern von Methoden

- Variablen mit Modifier final sind Konstanten.
 - Wert kann nach erster Zuweisung nicht mehr geändert werden

? Question

- Was meinen Sie?
 - Was bewirkt final für Klassen?
 - Was bewirkt final für Methoden?

-

-

5.1 Überlagern von Methoden

- Variablen mit Modifier final sind Konstanten.
 - Wert kann nach erster Zuweisung nicht mehr geändert werden

? Question

- Was meinen Sie?
 - Was bewirkt final für Klassen?
 - Was bewirkt final für Methoden?
- Klassen:
 - Klasse mit Modifier final kann nicht abgeleitet werden
 - Beispiel: Klasse String
- Methoden:
 - Methode mit Modifier final kann nicht in Subklasse überlagert werden

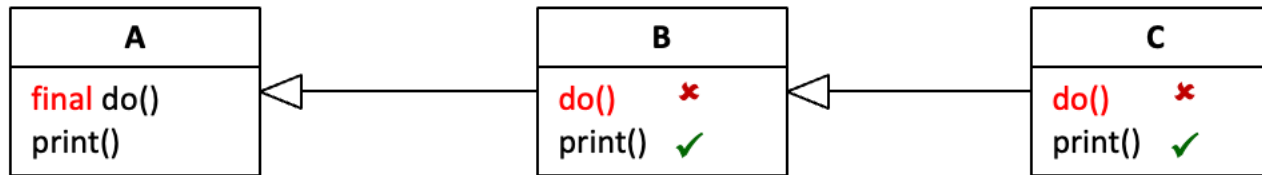
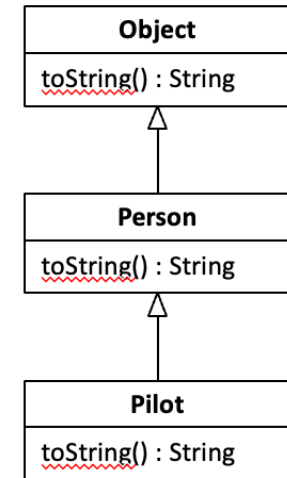


Figure 2: final-Schlüsselwort in der Vererbung

5. Überlagern



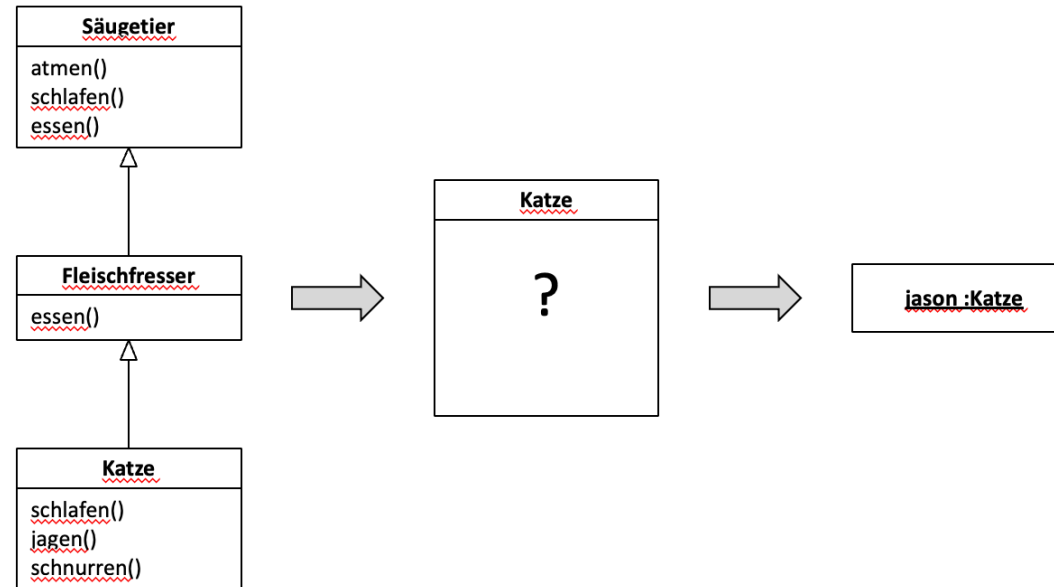
- Überlagern von Methoden:
 - Von der Basisklasse geerbte Methoden dürfen neu definiert werden.
 - Begriffe: Überlagern (oder auch Überschreiben)
 - Beim Aufruf wird die überlagernde Methode („neueste Version“) ausgeführt
 - Aufruf der verdeckten Methode `name()` der Basisklasse über `super.name()`
 - Modifier `final` unterbindet Überlagern in Subklassen
- Überlagern von Attributen:

5.1 Überlagern von Methoden

- Abgeleitete Klasse kann auf gleiche Weise Variablen der Basisklasse überlagern

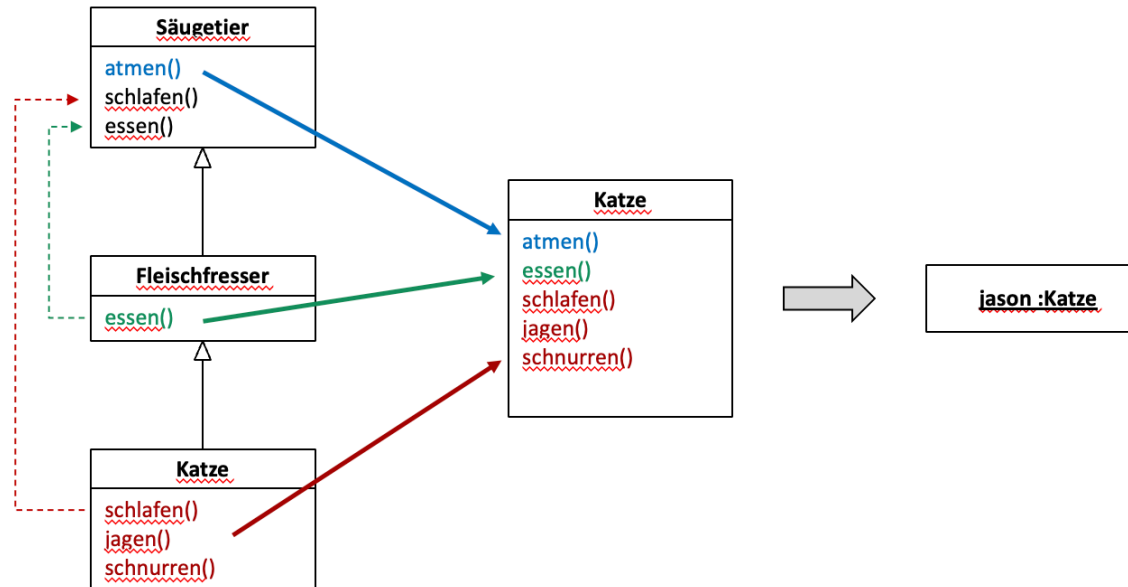
? Question

- Welche Methoden hat Jason, die Katze?
- Aus welchen Klassen stammt jeweils die Methoden-Definition?



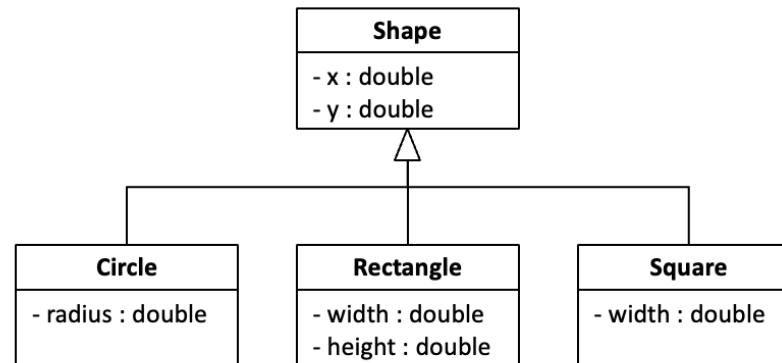
? Question

- Welche Methoden hat Jason, die Katze?
- Aus welchen Klassen stammt jeweils die Methoden-Definition?



Task 7

- Ergänzen Sie Methoden `getArea()` zur Bestimmung der Fläche eines Objekts.
- Erstellen Sie folgendes ausführbares Programm:
 - Speichert je ein Objekt `Circle`, `Rectangle` und `Square` in einer gemeinsamen Liste
 - Bestimmt Summe der Flächeninhalte aus dieser Liste

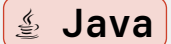


5.1 Überlagern von Methoden

5. Überlagern

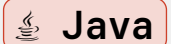
Klasse Circle:

```
1  public double getArea() {  
2      return Math.PI * radius * radius;  
3  }
```



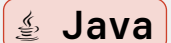
Klasse Rectangle:

```
1  public double getArea() {  
2      return width * height;  
3  }
```



Klasse Square:

```
1  public double getArea() {  
2      return width * width;  
3  }
```

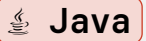


5.1 Überlagern von Methoden

5. Überlagern

- Ausführbares Programm:

```
1  public static void main(String[] args) {
2      ArrayList<Shape> shapes = new ArrayList<Shape>();
3      shapes.add(new Circle(2.0, 3.0, 1.0));
4      shapes.add(new Rectangle(-1.0, 0.0, 3.5, 4.0));
5      shapes.add(new Square(0.0, 0.0, 2.5));
6
7      double sumArea = 0.0;
8      for (Shape shape : shapes) {
9          sumArea += shape.getArea();
10     }
11
12     System.out.println("Overall area of shapes = " + sumArea);
13 }
```



- Methode `getArea()` muss auch in `Shape` implementiert sein
- Wird allerdings nicht verwendet, sondern von Subklassen überlagert: Sehr unschön!
- Wir werden später eine elegantere Lösung kennenlernen.

6. Polymorphism

- Die gute Nachricht:
 - Nur ein neuer Begriff, ansonsten ist alles bereits bekannt
 - Nein, wirklich. Ganz ehrlich. Echt wahr ...
- Polymorphismus:
 - Wortlaut: „Vielgestaltigkeit“
 - Methoden mit gleichem Namen können mehrere Gestalten annehmen.
 - Sprich: Mehrere Implementierungen von Methoden mit gleichem Namen
 - Typische Eigenschaft objektorientierter Sprachen

? Question

- Wo ist uns das bereits begegnet?

? Question

- Wo ist uns das bereits begegnet?
- Methoden gleichen Namens in derselben Klasse: Überladen
- Methoden gleichen Namens in Vererbungslinie: Überlagern (auch: Überschreiben)

6.1 Polymorphismus

- Überladen (Overloading):
 - Methoden in Klasse haben gleichen Namen
 - Müssen unterschiedliche Signatur haben (d.h. unterschiedliche Parametertypen)

MathFunctions
<code>max(int, int) : int</code> <code>max(int, int, int) : int</code> <code>max(double, double): double</code> <code>max(double[]): double</code>

Figure 6: Klasse MathFunctions

6. Polymorphismus

- Überlagern / Überschreiben (Overriding):
 - Methode in Vererbungslinie haben gleichen Namen
 - Müssen gleiche Signatur haben (d.h. gleichen Namen und Parametertypen)

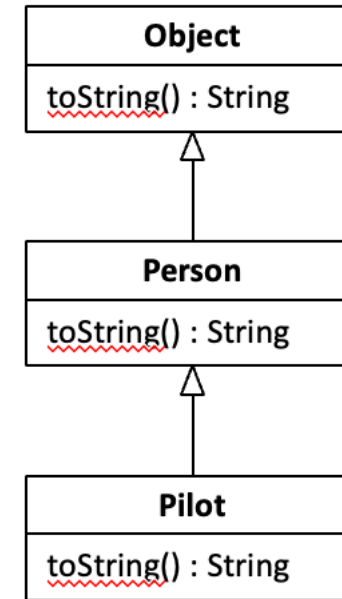


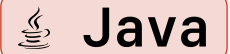
Figure 7: Vererbung von `toString()`

7. Vergleich von Objekten

? Question

- Klasse Point beinhalte die Variablen x und y
- Was wird ausgegeben?

```
1  Point a = new Point(1, 2);  
2  Point b = new Point(7, 3);  
3  System.out.println(a == b);
```



? Question

- Und nun?

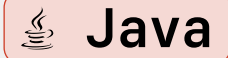
```
1  Point a = new Point(1, 2);  
2  Point b = new Point(1, 2);  
3  System.out.println(a == b);
```

**Java**

? Question

- Und nun?

```
1  Point a = new Point(1, 2);  
2  Point b = a;  
3  System.out.println(a == b);
```



7.1 Vergleichsoperator (==)

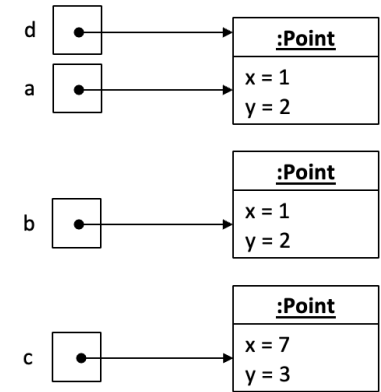
7. Vergleich von Objekten

- Vergleichsoperator vergleicht, ob Variablen denselben Inhalt haben
 - Inhalt ist jeweils Referenz auf ein Objekt
 - Vergleich nur dann wahr (true), wenn Variablen dasselbe Objekt referenzieren



Example

```
1 Point a = new Point(1, 2);  
2 Point b = new Point(1, 2);  
3 Point c = new Point(7, 3);  
4 Point d = a;
```



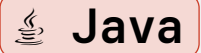
✗ Error

- `a == b`: Verschiedene Objekte (mit gleichen Werten)
- `a == c`: Verschiedene Objekte (und Werte)

✓ Success

- `a == d`: Dasselbe Objekt: gleiche Referenz

```
1  public boolean equals(Object obj) {  
2      // Methodenrumpf  
3      // Rückgabe eines Wertes vom Typ boolean  
4  }
```



- Vergleich, ob alle Variablen zweier referenzierter Objekte gleiche Werte haben
- Methode ist bereits in Klasse Object definiert
- Überlagern in eigenen Klassen:
 - Klasse Object kann nicht wissen, welchen Variablen Sie in Subklassen hinzufügen

7.2 equals()-Methode

- Methode daher gegebenenfalls überlagern, um hinzugefügte Attribute zu vergleichen
- In IntelliJ IDEA ist das bequem über das Generate-Menü möglich.

? Question

- Gegeben sei Klasse Point mit x- und y-Koordinate
- Welches Ergebnis liefern die Vergleiche in der Tabelle?

Quelltext	<code>a == b</code>	<code><u>a.equals(b)</u></code>
<pre>Point a = new Point(10, 25); Point b = new Point(10, 2);</pre>	<div>?</div>	
<pre>Point a = new Point(10, 25); Point b = new Point(10, 25);</pre>	<div>?</div>	
<pre>Point a = new Point(10, 25); Point b = a;</pre>	<div>?</div>	

? Question

- Gegeben sei Klasse Point mit x- und y-Koordinate
- Welches Ergebnis liefern die Vergleiche in der Tabelle?

Quelltext	a == b	<u>a.equals(b)</u>
Point a = new Point(10, 25); Point b = new Point(10, 2);	<u>false</u>	<u>false</u>
Point a = new Point(10, 25); Point b = new Point(10, 25);	<u>false</u>	<u>true</u>
Point a = new Point(10, 25); Point b = a;	<u>true</u>	<u>true</u>

8. License Notice

8.1 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.