Object-Oriented Programming in Java

Lecture 5 - Inheritance

Emily Lucia Antosch

HAW Hamburg

14.08.2025

Contents

1. Introduction

1.1 Where Are We Currently?

- The last lecture was about class libraries
- You can now
  - use simple class libraries to manipulate strings or arrays
  - read all items from an array via a `foreach` loop
  - perform type conversions using wrapper classes,
  - perform simple mathematical calculations using the `Math` class.
- Today we continue with **Inheritance**.

1.1 Where Are We Currently?

1. Imperative Concepts
2. Classes and Objects
3. Class Library
4. **Inheritance**
5. Interfaces
6. Graphical User Interfaces
7. Exception Handling
8. Input and Output
9. Multithreading (Parallel Computing)

1.2 The Goal of This Chapter

- You create new data types by extending existing classes with additional properties, for example to avoid duplicated source code.
- You use visibility modifiers to protect the attributes of a class from direct external access.

2. Inheritance

2.1 Inheritance

- Class takes over (“inherits”) variables and methods of an existing class
- Goal: Reuse of existing classes
- Example and UML notation:
  - Class A exists
  - Class B is created and inherits from A
- Terms:
  - Class A: Superclass (base class, parent class)
  - Class B: Subclass (derived class, child class)
  - Inheritance: Derivation, English: inheritance

2.1 Inheritance

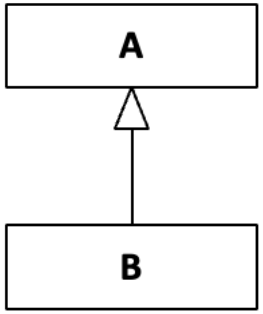


Figure 1: Simple example of inheritance

2.1 Inheritance

- Derivation from base class using `extends`:

```java
class ClassName extends BaseClass {
    Attributes
    Methods
    }
```

2.1 Inheritance

Example

Java

```java
class A {
        // ...
    }

    class B extends A {
        // ...
    }
```

2.1 Inheritance

Task 1

- Create the following classes:
  - Person: Objects contain the name
  - Pilot: Objects contain the name and previous flight hours
  - Executable class that creates a Pilot object and outputs the name

2.1 Inheritance

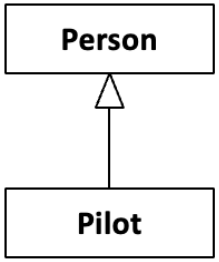


Figure 2: Pilot inherits from Person

2.1 Inheritance

Java

```java
public class Person {
        String name;
    }

    public class Pilot extends Person {
        int flightHours;
    }

    public class PilotDemo {
        public static void main(String[] args) {
            Pilot pilot = new Pilot();

            pilot.name = "Lukas Luft";
            pilot.flightHours = 1482;
            System.out.println("Name: " + pilot.name);
        }
    }
```

2.1 Inheritance

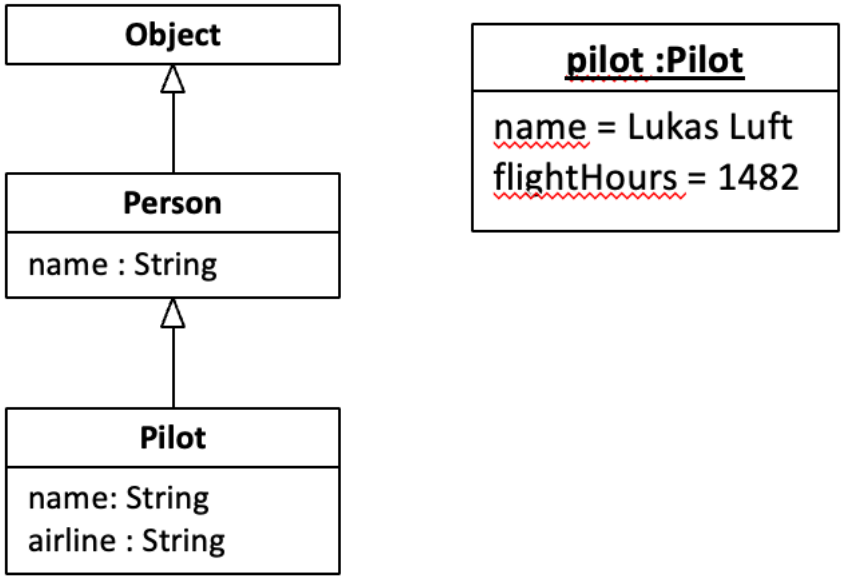
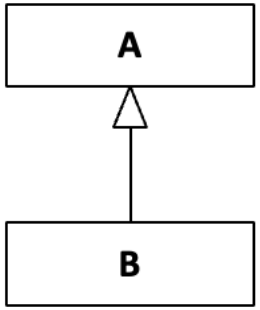


Figure 3: Inheritance of attributes

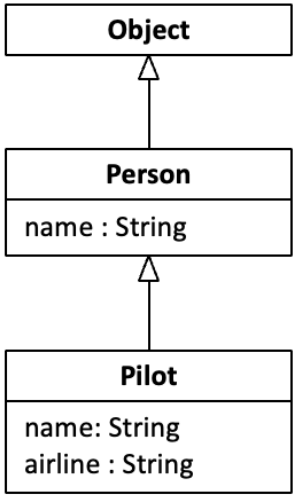
2.1 Inheritance

- Class B can add new variables and methods



- Terms:
  - Specialization: Class B is more specialized than Class A
  - Generalization: Class A is more general than Class B

- Example:
  - Class Pilot has inherited from Person and added flightHours
  - A Pilot is a Person, i.e. Person is more general than Pilot.



2.1 Inheritance

Java

```java
public class Person {
    String name;
}

public class Pilot extends Person {
    int flightHours;
}
```

2.1 Inheritance

- Data encapsulation (information hiding): Variables protected from external access
- Restrictions on access to classes, variables and methods through modifiers
- Mental model: “Visibility” (i.e. is element visible or known?)

2.1 Inheritance

Modifier	UML	Visibility	Class	Attributes	Methods
`public`	+	All Classes	x	x	x
`protected`	#	Subclasses, classes of the package		x	x
`private`	-	Within the class		x	x
`<none>`	~	Classes of the package	x	x	x

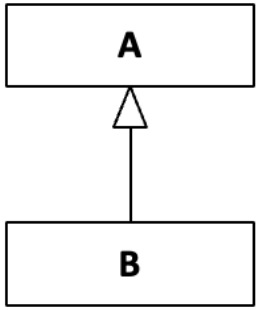
Table 1: Formats and Flags

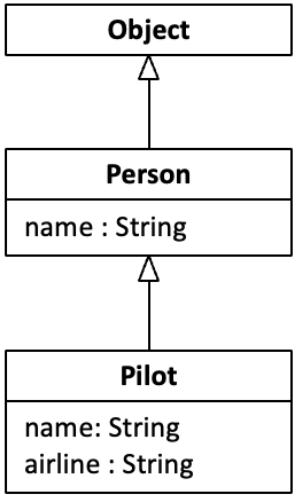
? Question

Is `private` allowed as a modifier for constructors?

2.1 Inheritance

- Class B inherits all visible variables and methods of Class A
- Class B possesses variables and methods from A and can use them (as if they were defined in Class B)
- Example:
  - Object of Class Pilot uses variable name from base class Person





2.1 Inheritance

Java

```java
public class PilotDemo {
    public static void main(String[] args) {
        Pilot pilot = new Pilot();

        pilot.name = "Lukas Luft";
        pilot.flightHours = 1482;
        System.out.println("Name: " + pilot.name);
    }
}
```

2.1 Inheritance

? Question

- What do you think, which components of a class are not inherited?

- Not passed to derived class:
  - Constructors and destructors
  - Class variables and class methods (modifier static)
  - Private variables and methods (modifier private)

2.1 Inheritance

- Notes:
  - Static elements never inherited, as they are bound to a class and not to a concrete object
  - Private elements are present in subclass, but it cannot directly access them

2.1 Inheritance

- Subclasses can be further inherited.
- Any number of subclasses can be derived from one class.
- However, inheriting from multiple base classes is not possible (multiple inheritance)

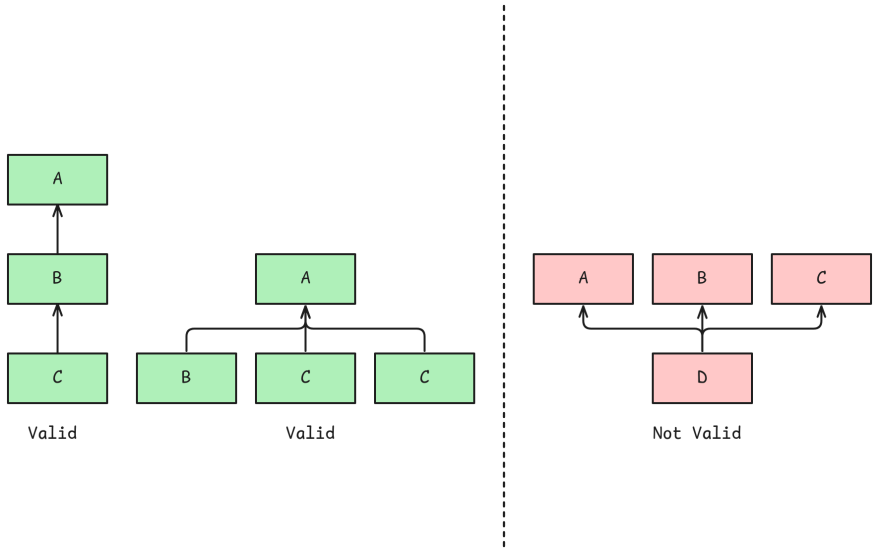


Figure 6: Possible structures for inheritance

2.1 Inheritance

? Question

- What do you think, which base class does Person have?

```java
public class Person {
    String name;
}
```

- You couldn't have known this before:
  - In Java a class Object is defined.
  - No base class specified. Implicitly derived from Object (extends Object)

2.1 Inheritance

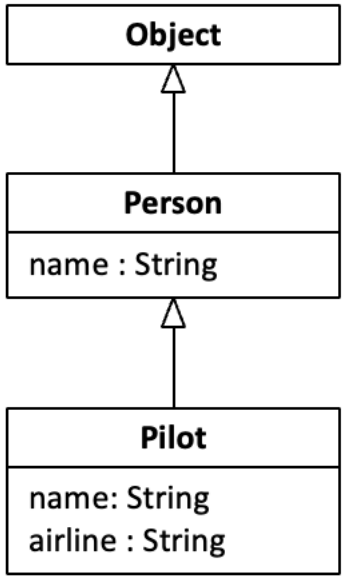


Figure 7: Object as superclass of Person

2.1 Inheritance

- Important consequence:
  - Object is the base class of every inheritance hierarchy

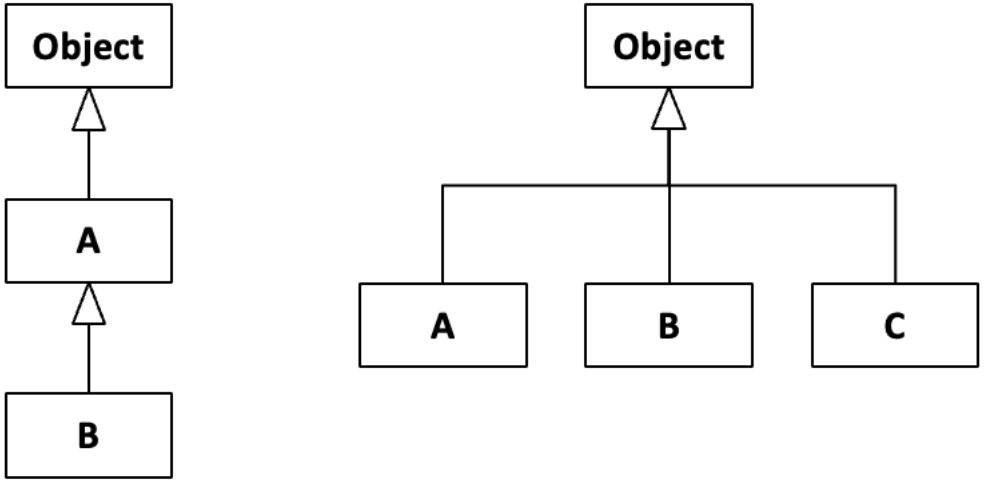


Figure 8: Object as part of every inheritance

2.1 Inheritance

? Question

- What do you think?
  - How many classes have no base class?
  - How many classes have more than one direct base class?

2.1 Inheritance

- Important consequence:
- Every class inherits the methods defined in Object (e.g. toString())
- Example:

```java
public class Person {
  String name;
}

public class ObjectDemo {
  public static void main(String[] args) {
      Person person = new Person();

      person.name = "Lukas Luft";
      System.out.println(person.toString());
  }
}
```

2.1 Inheritance

Task 2

- Implement classes for geometric objects Circle, Rectangle and Square.
- Use only public variables for now.
- Don’t implement any methods for now.

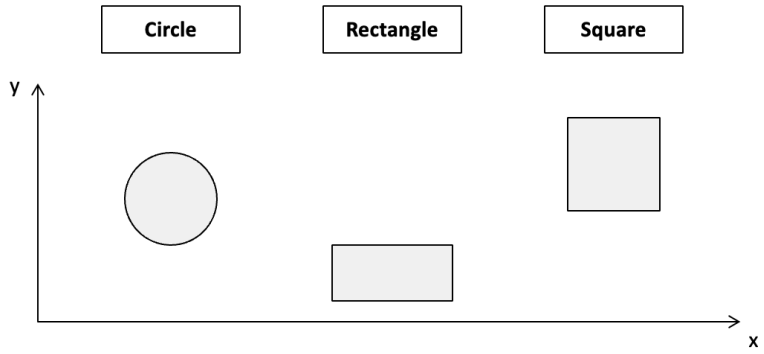


Figure 9: Geometric shapes as objects

2.1 Inheritance


```java
public class Circle {
    public double x, y;
    public double radius;
}

public class Rectangle {
    public double x, y;
    public double width, height;
}

public class Square {
    public double x, y;
    public double width;
}
```

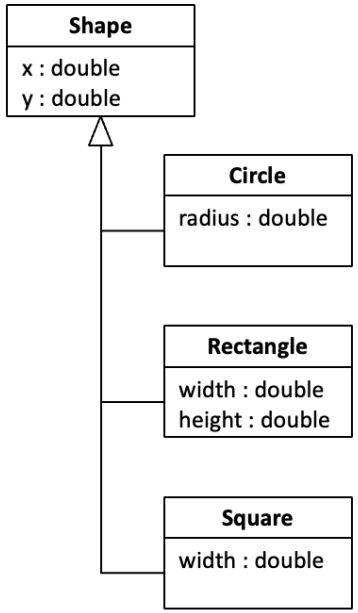
Task 3

Now create a common base class!

2.1 Inheritance

Java

```java
public class Shape {
        public double x, y;
    }

    public class Circle extends Shape {
        public double radius;
    }

    public class Rectangle extends Shape {
        public double width, height;
    }

    public class Square extends Shape {
        public double width;
    }
```



2.1 Inheritance

Question

- Which variables are declared in the respective classes?

Task 4

- Add a constructor for the Circle class!

2.1 Inheritance

Java

```java
public class Shape {
    public double x, y;
}

public class Circle extends Shape {
    public double radius;

    public Circle(double x, double y, double radius) {
        this.x = x;
        this.y = y;
        this.radius = radius;
    }
}
```

Memorize

- Note: Variables x and y of the base class are used like “own” variables
- Hide the variables of the Shape class through the private modifier.

2.1 Inheritance

Java

```java
public class Shape {
        private double x, y;
    }

    public class Circle extends Shape {
        public double radius;

        public Circle(double x, double y, double radius) {
            this.x = x;
            this.y = y;
            this.radius = radius;
        }
    }
```

Error

- The variables x and y of the base class are not visible in Circle.
- Error: In the constructor of class Circle, x and y are unknown.

2.1 Inheritance

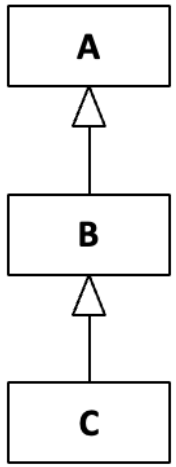
Java

```java
public class Shape {
        private double x, y;

        public void setX(double x) {
            this.x = x;
        }
        // Additionally getter and corresponding methods for y ...
    }

    public class Circle extends Shape {
        public double radius;

        public Circle(double x, double y, double radius) {
            setX(x);
            setY(y);
            this.radius = radius;
        }
    }
```

3. Object Instantiation

3.1 Object Creation

- Class C contains its own methods as well as methods from classes A and B.
- Class C contains its own variables as well as variables from classes A and B.



? Question

- What do you think?
  - How are methods of a new object of class C created?
  - How are variables of an object of class C created and initialized?

- Methods:
  - Are not created anew for each object, but are defined for the class
- Variables:
  - Start at the base class of the inheritance hierarchy
  - In each step, create and initialize variables of the corresponding (base) class
  - Initialization via constructor of the respective (base) class

3.1 Object Creation

- Variables for objects of class C:
  - Object contains the variables declared in class C
  - Additionally contains variables inherited from class B
  - These contain the variables inherited from class A

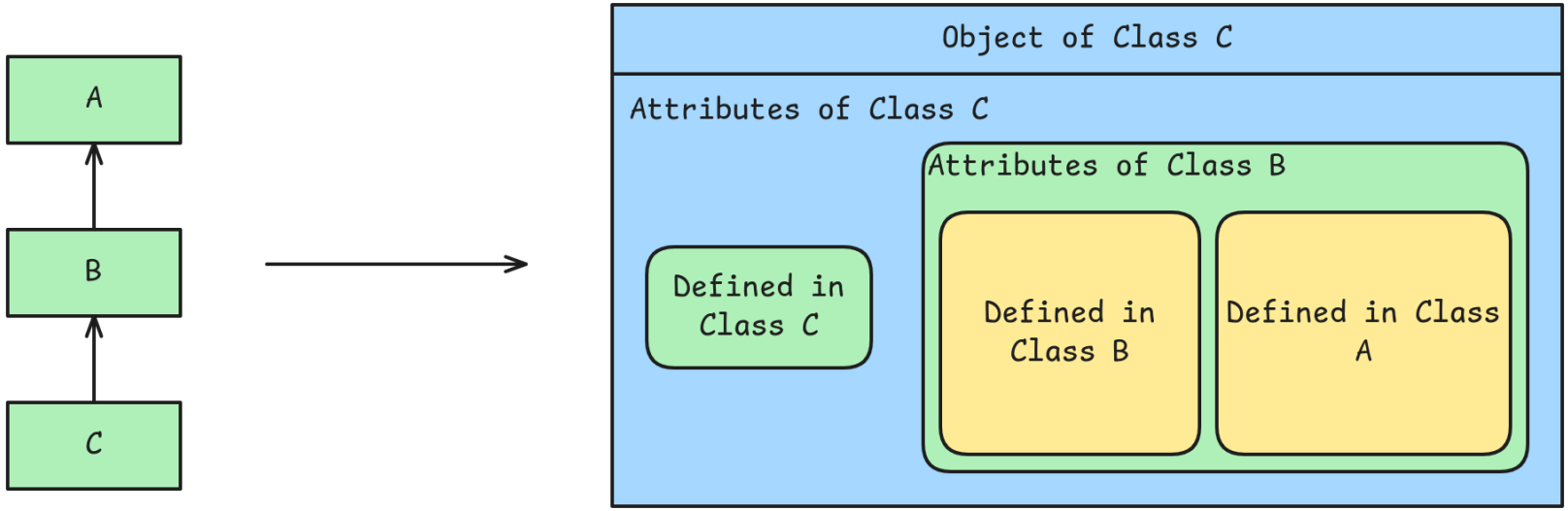


Figure 12: Composition of the object of class C

3.1 Object Creation

- New object of class C:
  - Traverse inheritance hierarchy upward:
  - Class C has base class B: Call to create variables from B
  - Class B has base class A: Call to create variables from A
- Create and initialize variables “from inside out” (constructor chaining):
  - Create variables from A and initialize via constructor A()
  - Create variables from B and initialize via constructor B()
  - Create variables from C and initialize via constructor C()

3.1 Object Creation

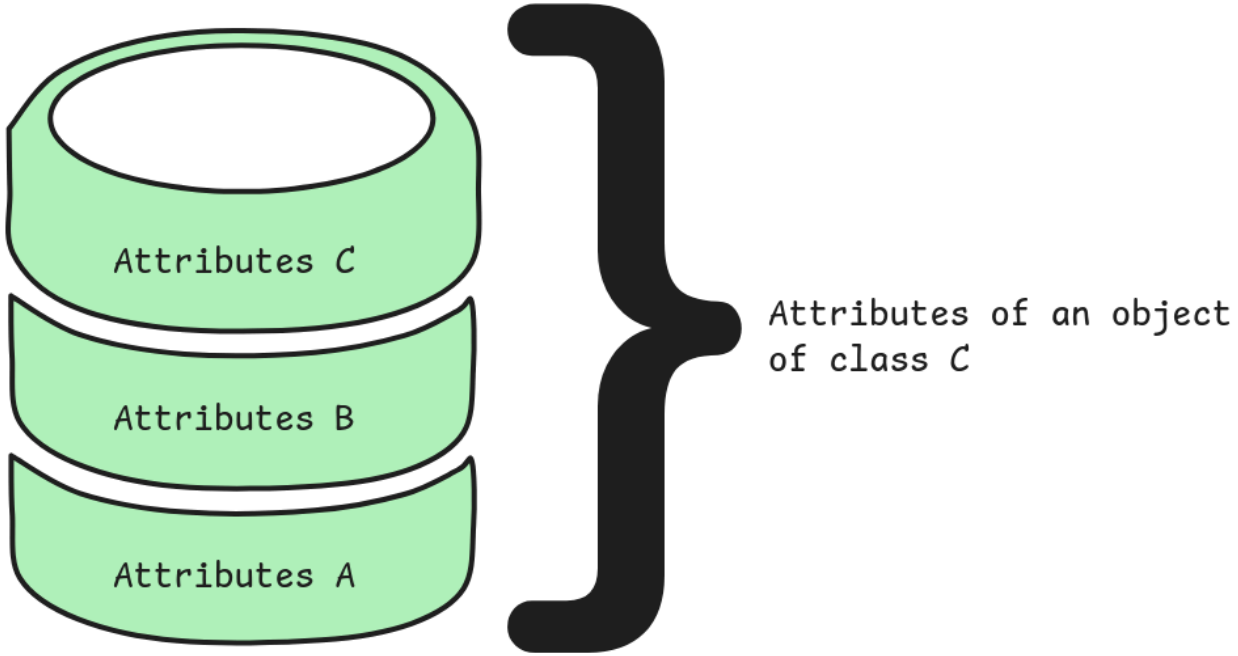
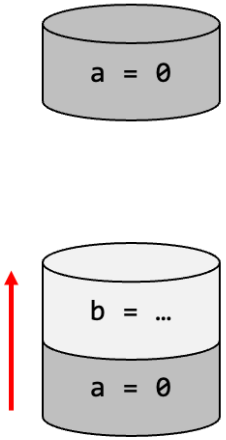


Figure 13: Variables of class C in memory

3.1 Object Creation

- Constructor of the base class:
  - Call via super() as first statement in constructor of derived class
  - If super(…) is missing, the default constructor of the base class is called.



Java

```java
public class A {
    double a;
    // Default constructor is automatically generated
}

public class B extends A {
    double b;

    public B(double b) {
        super();    // Call default constructor of class A
        this.b = b;
    }
}
```

3.1 Object Creation

- Do you remember?

Java

```java
public class Shape {
    private double x, y;
}

public class Circle extends Shape {
    public double radius;

    public Circle(double x, double y, double radius) {
        this.x = x;
        this.y = y;
        this.radius = radius;
    }
}
```

3.1 Object Creation

Task 5

- The variables x and y are unknown in Circle.
- Solve the problem by adding a constructor for the base class Shape.

3.1 Object Creation

Java

```java
public class Shape {
    private double x, y;

    public Shape(double x, double y) {
        this.x = x;
        this.y = y;
    }
}

public class Circle extends Shape {
    public double radius;

    public Circle(double x, double y, double radius) {
        super(x, y);  // Matching signature to the constructor of the base class!
        this.radius = radius;
    }
}
```

3.1 Object Creation

Task 6

- Protect all attributes through the private modifier.
- Create appropriate getters and setters if necessary.

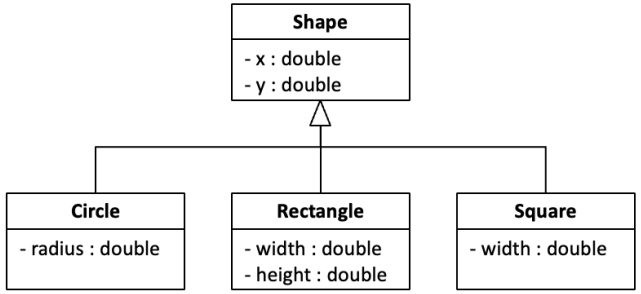


Figure 15: Structure of inheritance

3.1 Object Creation

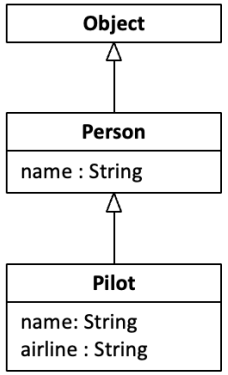
Java

```java
public class Circle extends Shape {
    private double radius;

    public Circle(double x, double y, double radius) {
        super(x, y);
        this.radius = radius;
    }

    public double getRadius() {
        return radius;
    }

    public void setRadius(double radius) {
        this.radius = radius;
    }
}
```

4. Referencing via Base Class

4.1 Base Class Reference

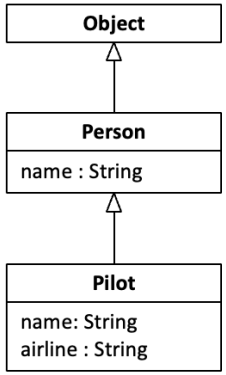
- Let's consider the following inheritance line:




```java
public class Person {
    String name;

    public Person(String name) {
        this.name = name;
    }
}

public class Pilot extends Person {
    String airline;

    public Pilot(String name, String airline) {
        super(name);
        this.airline = airline;
    }
}
```

4.1 Base Class Reference

- What do you think about the following program?




```java
public class ReferenceDemo {
  public static void main(String[] args) {
    Pilot pilot = new Pilot("Birgit", "Winglet Airways");
    Person personRef = pilot;
    Object objectRef = pilot;

  System.out.println(personRef.name);
  }
}
```

Memorize

There is only one object (with data type Pilot). Object is referenced via variables with other data types than Pilot

4.1 Base Class Reference

- Class Pilot inherits from class Person and extends it
- Pilot contains Person (“Pilot is a Person”) Referenceable as Person
- Object is not changed by this (i.e. object remains of type Pilot)!

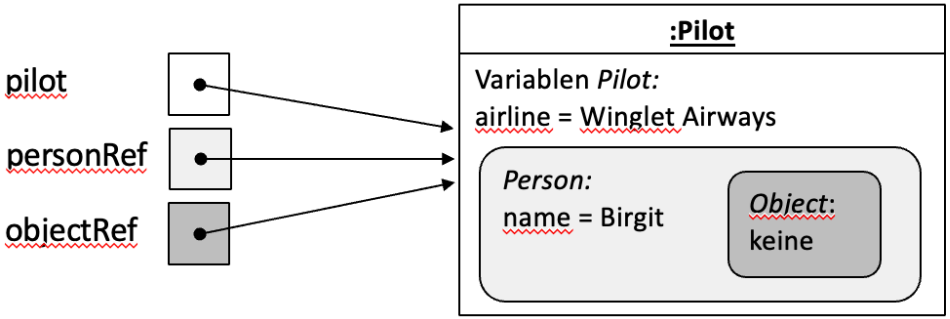


Figure 18: References to an object with base class

4.1 Base Class Reference

- In general:
  - Objects can be treated like objects of their base classes.
  - Objects referenceable via data types of their base classes
  - Reference variable can only access attributes and methods of its class

4.1 Base Class Reference

? Question

- Which accesses to attributes are allowed and which are not?

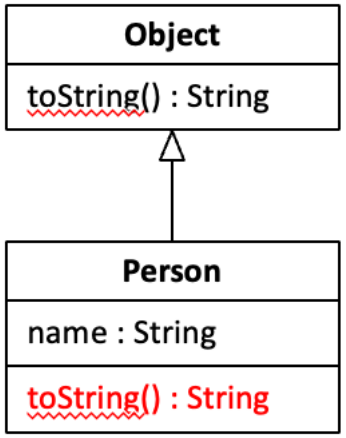
```java
public static void main(String[] args) {
    Pilot pilot = new Pilot("Birgit", "Winglet Airways");
    Person personRef = pilot;
    Object objectRef = pilot;

    System.out.println(personRef.name);
    System.out.println(personRef.airline);
    System.out.println(objectRef.name);
    System.out.println(objectRef.airline);
}
```

5. Overriding

5.1 Overriding Methods

- Class Person is derived from Object and therefore inherits toString() from Object and therefore defines another toString() method

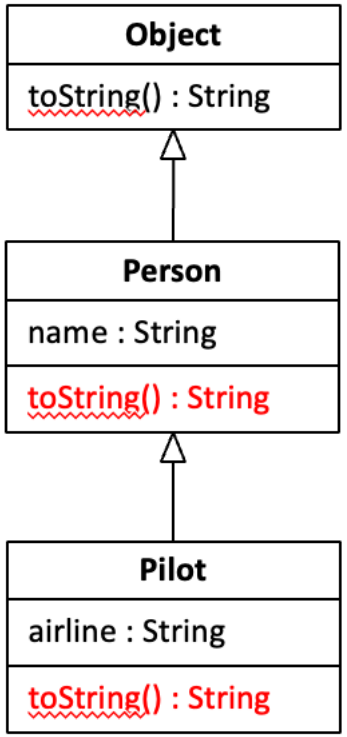
```java
public class Person {
    private String name;

    public Person(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public String toString() {
        return name;
    }
}
```



5.1 Overriding Methods

- Class Pilot:
  - Derived from Person and therefore inherits toString() from Person
  - Defines another toString() method

```java
public class Pilot extends Person {
    private String airline;

    public Pilot(String name, String airline) {
        super(name);
        this.airline = airline;
    }

    public String toString() {
        return String.format("%s (%s)", getName(), airline);
    }
}
```



5.1 Overriding Methods

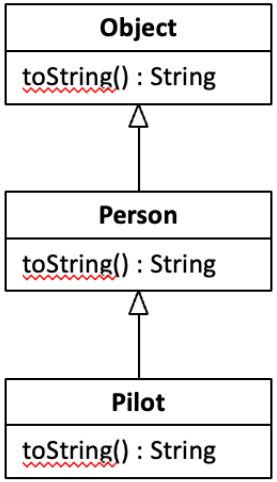
? Question

What will be output?

```java
public static void main(String[] args) {
        Person person = new Person("Birgit Janssen");
        System.out.println("person: " + person);
        System.out.println("person.toString(): " + person.toString());

        Pilot pilot = new Pilot("Jan Birgerson", "Winglet Airways");
        System.out.println("pilot.toString(): " + pilot.toString());
    }
```

5.1 Overriding Methods

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



! Memorize

- Respective method of the corresponding class, not the superclass(es), executed
- Term: Method of the superclass overridden by newly defined methods

5.1 Overriding Methods

? Question

- What will be output?

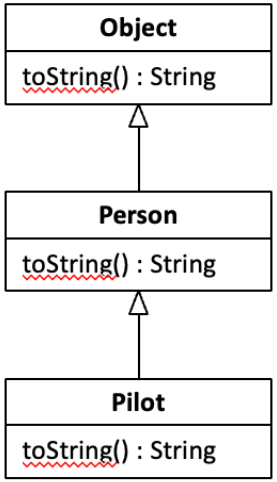
```java
public static void main(String[] args) {
    Pilot pilot = new Pilot("Jan Birgerson", "Winglet Airways");
    Object objectRef = pilot;
    Person personRef = pilot;

    System.out.println("objectRef: " + objectRef.toString());
    System.out.println("personRef: " + personRef.toString());
}
```

5.1 Overriding Methods

? Question

- What will be output?

```java
public static void main(String[] args) {
    Pilot pilot = new Pilot("Jan Birgerson", "Winglet Airways");
    Object objectRef = pilot;
    Person personRef = pilot;

    System.out.println("objectRef: " + objectRef.toString());
    System.out.println("personRef: " + personRef.toString());
}
```



- Output:
  - `objectRef`: Jan Birgerson (Winglet Airways)
  - `personRef`: Jan Birgerson (Winglet Airways)

5.1 Overriding Methods

! Memorize

Method of the corresponding class executed, even when referenced via superclass(es)

5.1 Overriding Methods

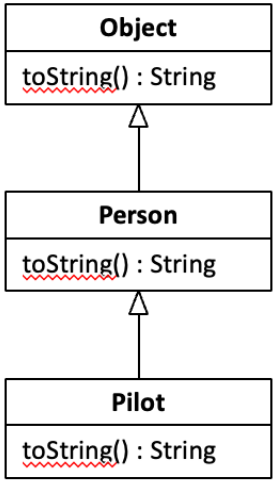
- Access to overridden methods of the base class via reference super
- Example:

```java
public class Pilot extends Person {
    // Instance variable, constructor, toString() ...

    public String toStringOfSuperClass() {
        return super.toString();
    }
}

public static void main(String[] args) {
    Pilot pilot = new Pilot("Jan Birgerson", "Winglet Airways");
    System.out.println("Pilot.toString(): " + pilot.toString());
    System.out.println("super.toString(): " + pilot.toStringOfSuperClass());
}
```

5.1 Overriding Methods

- Access to overridden methods of the base class via reference super
- Example:

```java
public class Pilot extends Person {
    // Instance variable, constructor, toString() ...

    public String toStringOfSuperClass() {
        return super.toString();
    }
}

public static void main(String[] args) {
    Pilot pilot = new Pilot("Jan Birgerson", "Winglet Airways");
    System.out.println("Pilot.toString(): " + pilot.toString());
    System.out.println("super.toString(): " + pilot.toStringOfSuperClass());
}
```



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

5.1 Overriding Methods

- Variables with modifier final are constants.
  - Value cannot be changed after first assignment

? Question

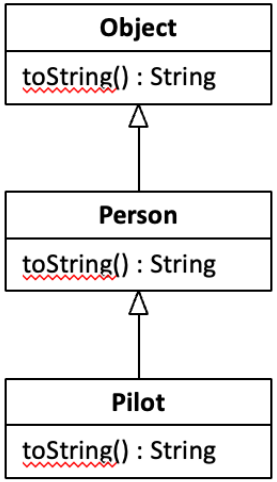
- What do you think?
  - What does final do for classes?
  - What does final do for methods?

-

-

5.1 Overriding Methods

- Variables with modifier final are constants.
  - Value cannot be changed after first assignment



? Question

- What do you think?
  - What does final do for classes?
  - What does final do for methods?

- Classes:
  - Class with modifier final cannot be derived
  - Example: String class
- Methods:
  - Method with modifier final cannot be overridden in subclass

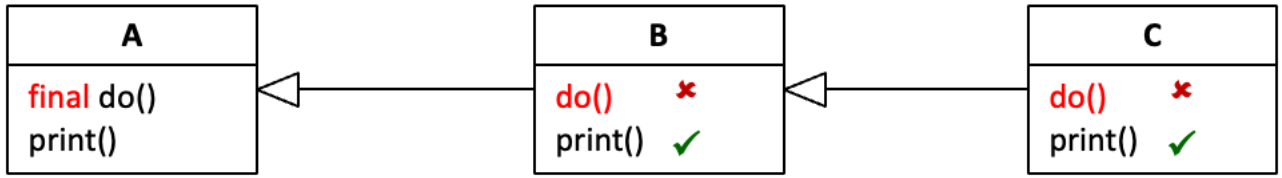


Figure 2: `final` keyword in inheritance

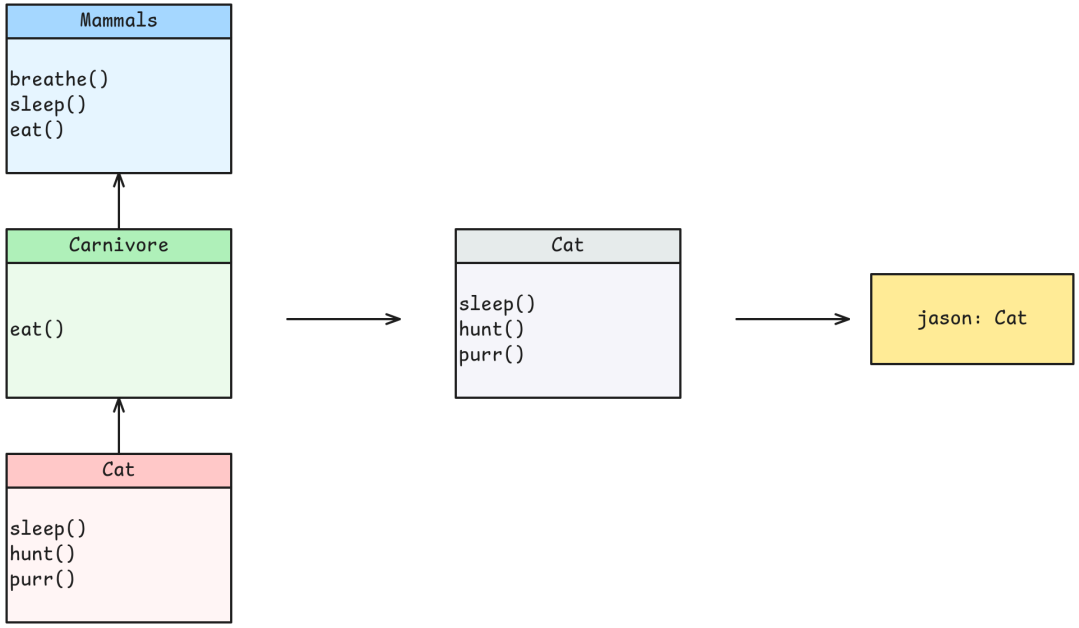
5.1 Overriding Methods

- Overriding methods:
  - Methods inherited from the base class may be redefined.
  - Terms: Overriding (or also overwriting)
  - When called, the overriding method (“newest version”) is executed
  - Call the hidden method `name()` of the base class via `super.name()`
  - Modifier `final` prevents overriding in subclasses
- Overriding attributes:
  - Derived class can override variables of the base class in the same way

5.1 Overriding Methods

? Question

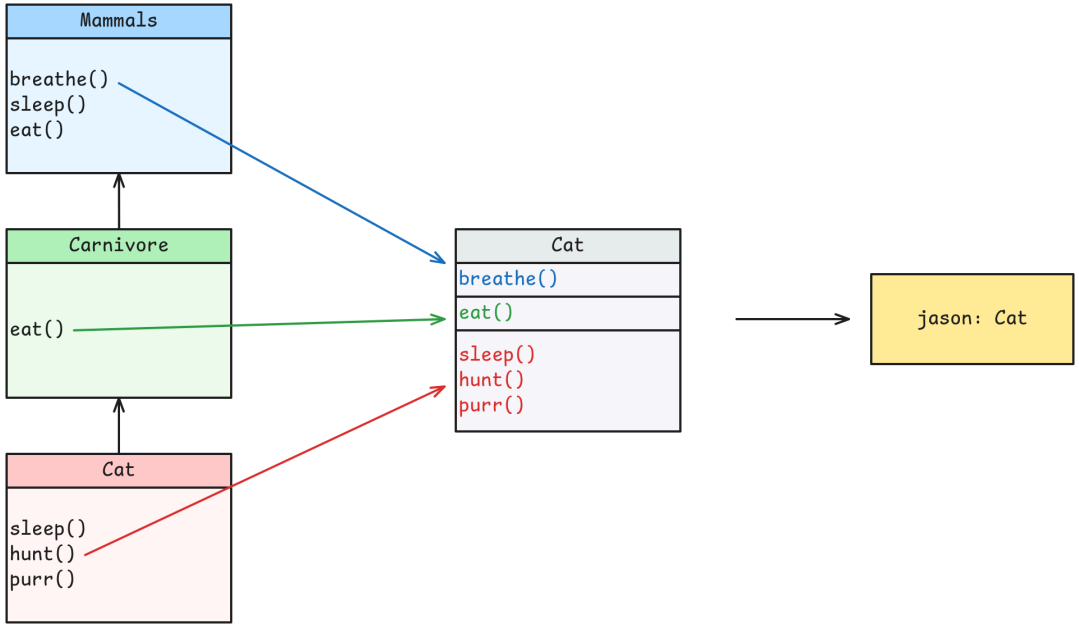
- Which methods does Jason, the cat, have?
- From which classes does each method definition come?



5.1 Overriding Methods

? Question

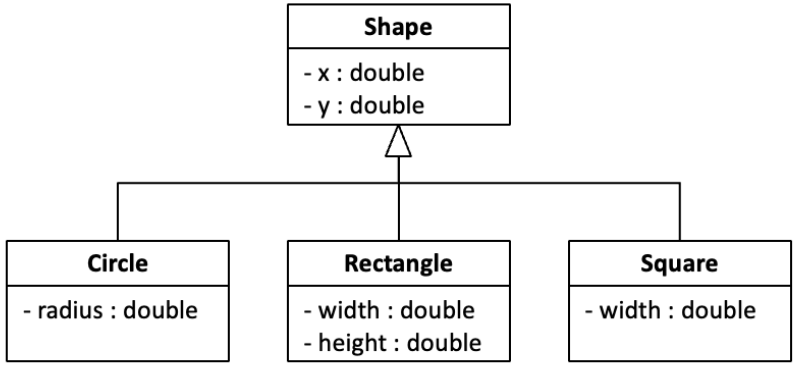
- Which methods does Jason, the cat, have?
- From which classes does each method definition come?



5.1 Overriding Methods

Task 7

- Add methods `getArea()` to determine the area of an object.
- Create the following executable program:
  - Stores one object each `Circle`, `Rectangle` and `Square` in a common list
  - Determines sum of areas from this list



5.1 Overriding Methods

Klasse `Circle`:

```java
public double getArea() {
    return Math.PI * radius * radius;
}
```

Klasse `Rectangle`:

```java
public double getArea() {
    return width * height;
}
```

Klasse `Square`:

```java
public double getArea() {
    return width * width;
}
```

5.1 Overriding Methods

- Executable program:

```java
public static void main(String[] args) {
    ArrayList<Shape> shapes = new ArrayList<Shape>();
    shapes.add(new Circle(2.0, 3.0, 1.0));
    shapes.add(new Rectangle(-1.0, 0.0, 3.5, 4.0));
    shapes.add(new Square(0.0, 0.0, 2.5));

    double sumArea = 0.0;
    for (Shape shape : shapes) {
        sumArea += shape.getArea();
    }

    System.out.println("Overall area of shapes = " + sumArea);
}
```

- Method `getArea()` must also be implemented in `Shape`
- However, it is not used, but overridden by subclasses: Very ugly!
- We will learn a more elegant solution later.

6. Polymorphismus

6.1 Polymorphismus

- The good news:
  - Only a new term, otherwise everything is already known
  - No, really. Honestly. Really true …
- Polymorphism:
  - Literal meaning: “Many forms”
  - Methods with the same name can take multiple forms.
  - In other words: Multiple implementations of methods with the same name
  - Typical characteristic of object-oriented languages

6.1 Polymorphismus

? Question

- Where have we already encountered this?

6.1 Polymorphismus

? Question

- Where have we already encountered this?

- Methods with the same name in the same class: Overloading
- Methods with the same name in inheritance line: Overriding (also: Overwriting)

6.1 Polymorphismus

- Overloading:
  - Methods in class have the same name
  - Must have different signatures (i.e. different parameter types)

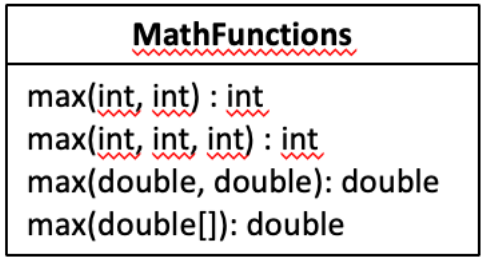


Figure 6: Class MathFunctions

- Overriding / Overwriting:
  - Methods in inheritance line have the same name
  - Must have the same signature (i.e. same name and parameter types)

6.1 Polymorphismus

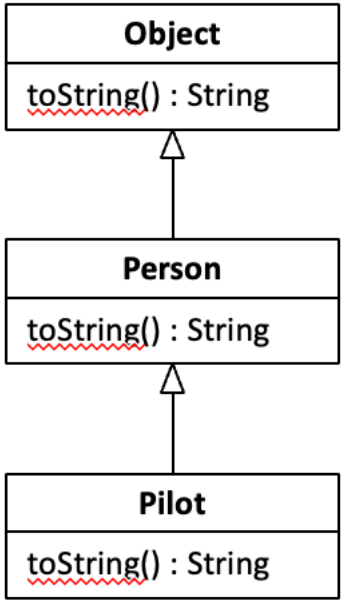


Figure 7: Inheritance of `toString()`

7. Comparing Objects

7.1 Comparison Operator (==)

? Question

- Class Point contains the variables x and y
- What will be output?

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

7.1 Comparison Operator (==)

? Question

- And now?

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

7.1 Comparison Operator (==)

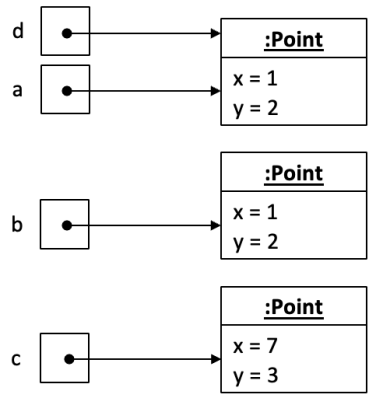
? Question

- Und nun?

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

7.1 Comparison Operator (==)

- Comparison operator compares whether variables have the same content
  - Content is respectively a reference to an object
  - Comparison only true when variables reference the same object



Example

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

Error

- `a == b`: Different objects (with same values)
- `a == c`: Different objects (and values)

Success

- `a == d`: Same object: same reference

7.2 `equals()`-Methode

```java
public boolean equals(Object obj) {
    // Method body
    // Return a value of type boolean
}
```

- Comparison whether all variables of two referenced objects have the same values
- Method is already defined in class Object
- Overriding in own classes:
  - Class Object cannot know which variables you add in subclasses
  - Therefore override method if necessary to compare added attributes
  - In IntelliJ IDEA this is conveniently possible via the Generate menu.

7.2 `equals()`-Methode

? Question

- Given class Point with x and y coordinates
- What result do the comparisons in the table provide?

Quelltext	a == b	a.equals(b)
`Point a = new Point(10, 25);` `Point b = new Point(10, 2);`	?	
`Point a = new Point(10, 25);` `Point b = new Point(10, 25);`	?	
`Point a = new Point(10, 25);` `Point b = a;`	?	

7.2 `equals()`-Methode

? Question

- Given class Point with x and y coordinates
- What result do the comparisons in the table provide?

Quelltext	a == b	a.equals(b)
`Point a = new Point(10, 25);` `Point b = new Point(10, 2);`	false	false
`Point a = new Point(10, 25);` `Point b = new Point(10, 25);`	false	true
`Point a = new Point(10, 25);` `Point b = a;`	true	true

8. License Notice

8.1 Attribution

- 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.