# Recap

- Object-Oriented Programming
  - Dealing with complexity by structuring your code
  - Classes and objects
- Classes
  - Unit of code to define some type of object
  - Contains fields (= variables, data) and methods (= behaviour)
- Objects
  - Concrete individuals of a certain class

```java
public class Horse {
  // the fields/variables of a class to store data about an instance
  String color;
  String name;
  int currentSpeed;

  // constructor to define what happens when a new object is created
  public Horse(String name) {
    this.name = name; // "this" to distinguish field and local variable
  }

  // methods of the class to define their behaviour
  public Horse mate(Horse partner) {
    // two horses meet and make a new horse
  }

  public static void main(String[] args) {
    // create an instance of type horse
    Horse h1 = new Horse("Joe");
    // create a second instance of type horse
    Horse h2 = new Horse("Jane");
  }
}
```

# demo

The class AsciiImage

# Ausschreibung für 2 Stellen als Studentische Hilfskraft

Computational LiteraryStudies

Gefördert durch DFG Deutsche Forschungsgemeinschaft

Im Rahmen des von der DFG geförderten Projekts CompAnno: Comparative Annotation to Explore and Explain Text Similarities im Schwerpunktprogramm *Computational Literary Studies* sind an der Universität zu Köln zwei Stellen als Studentische Hilfskraft für maximal drei Jahre zu besetzen.

## Projekt- und Aufgabenbeschreibung

Das von Prof. Dr. Julia Nantke (Universität Hamburg) und Prof. Dr. Nils Reiter (Universität zu Köln) kooperativ geführte Projekt untersucht Figurenbeschreibungen in deutschsprachigen Romanen. Teil des Projektes ist das Segmentieren von Textstellen, die Figurenbeschreibungen enthalten, um die Beschreibungen in den erzeugten Segmenten in einem späteren Schritt miteinander vergleichen zu können.

Hauptbestandteil Ihrer Aufgabe wird sein, die ausgewählten Texte nach bestehenden Annotationsrichtlinien zu segmentieren. Gegebenenfalls können weitere kleinere Annotationsarbeiten innerhalb des Projektes, wie etwa das Testen der Annotation von Segmentvergleichen, anfallen.

Die Arbeitszeit beträgt 40 Stunden / Monat und wird gemäß der Vergütungsliste für SHKs der Universität zu Köln vergütet.

## Anforderungen

- Bereitschaft, sich in das Thema, die Annotationsrichtlinien und das Annotationstool einzuarbeiten
- Bereitschaft, die im Projekt ausgewählten Romane zu lesen
- Bereitschaft, regelmäßig an Treffen vor Ort und online teilzunehmen
- Vorerfahrungen mit Annotationsarbeit (z.B. aus einem Kurs) ist ein Plus

## Bewerbung

Ihre Bewerbung richten Sie bitte bis zum **31. Januar 2024** als PDF-Dokument (max. 1 Seite) an janis.pagel@uni-koeln.de

Ihre Bewerbung soll in Kürze enthalten:

- Ihren CV
- Eine Begründung Ihres Interesses am Projekt

Die Universität zu Köln fördert Chancengerechtigkeit und Vielfalt. Frauen sind besonders zur Bewerbung eingeladen und werden nach Maßgabe des LGG NRW bevorzugt berücksichtigt. Bewerbungen von Menschen mit Schwerbehinderung und ihnen Gleichgestellten sind ebenfalls ausdrücklich erwünscht.

## Kontakt

Janis Pagel, M.Sc.
janis.pagel@uni-koeln.de
Universitätsstraße 22
50931 Köln

UNIVERSITÄT ZU KÖLN

Contact:
janis.pagel@uni-koeln.de

UNIVERSITÄT
ZU KÖLN

# Session 9: Methods and Inheritance

## Softwaretechnologie: Java 1

Nils Reiter
nils.reiter@uni-koeln.de

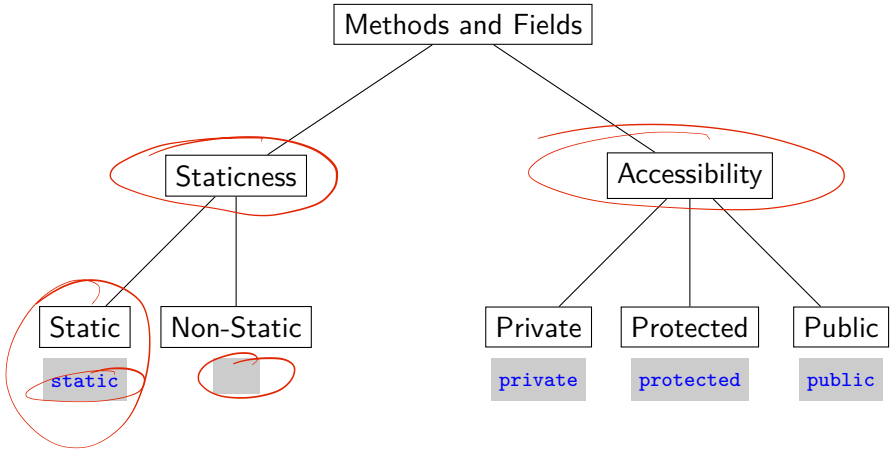
December 20, 2022

INSTITUT FÜR
DIGITAL HUMANITIES
UNIVERSITÄT ZU KÖLN

Section 1

# Methods

# Introduction



# Staticness

Non-static

- Methods can only be used with an object of the class in which they are defined
  - E.g., in order to call method `mate(Horse)`, one needs an object of type Horse
- Default behaviour (unmarked methods are non-static)
- Also applies to fields

Static

- Methods can be used without an object
  - E.g., marking a species as endangered is something for the class, not for instances of it
- Java keyword `static`

```java
public class Horse {
  // the fields/variables of a class to store data about an instance
  String name;

  // boolean field to store wether the species is extinct in the wild
  static boolean extinctInTheWild;

  public Horse mate(Horse partner) {
    // two horses meet and make a new horse
  }

  public static boolean isExtinctInTheWild() {
    return extinctInTheWild;
  }

  public static void main(String[] args) {
    Horse h1 = new Horse();
    Horse h2 = new Horse();
    Horse h3 = h1.mate(h2);

    if (Horse.isExtinctInTheWild()) {
      // do something
    }
  }
}
```

# Accessibility

- Public access – `public`
  - Method/field can be accessed from anywhere
- Protected access – `protected`
  - Method/field can only be accessed from within the same package
  - If no access is specified, it's protected
- Private access – `private`
  - Method/field can only be accessed from within the same class

## Accessibility

- Public access – `public`
  - Method/field can be accessed from anywhere
- Protected access – `protected`
  - Method/field can only be accessed from within the same package
  - If no access is specified, it's protected
- Private access – `private`
  - Method/field can only be accessed from within the same class

### Why?

- Modularization is important for dealing with complexity
- A complex program consists of many small parts that are not as complex
- Small parts are only maintainable if they have restricted interfaces
- Access restrictions can enfore that

# demo

Horse with static and private fields/methods

Section 2

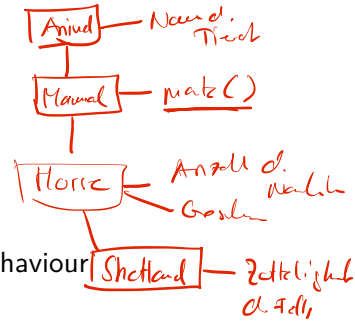
# Inheritance

# Introduction

Inheritance – “Vererbung”

- Important concept in object-oriented programming
- Classes represent kinds of things, because they show similar behaviour
  - Not all kinds are totally unique
  - Many kinds share certain properties
- E.g. Donkeys move in a similar way as horses do and both are mammals etc.

# Introduction




Inheritance – "Vererbung"

- Important concept in object-oriented programming
- Classes represent kinds of things, because they show similar behaviour
  - Not all kinds are totally unique
  - Many kinds share certain properties
- E.g. Donkeys move in a similar way as horses do and both are mammals etc.
- Inheritance allows us to model this
- Many domains have hierarchical structures
  - E.g., animal species, companies, kitchen equipment

```
Shetland p = new Shetland();
p.mate(...);
```

# Class Inheritance

- A class inherits from another class
- New keyword: `extends`, used in the class declaration:

```
public class Horse extends Animal { ... }
```

- Horse: sub class
- Animal: super class

# Class Inheritance

## Meaning

- No change in accessibility/visibility rules
  - private fields/methods still not visible, protected only within the same package etc.
- Objects of sub class can execute methods defined in super class
  - E.g., the class Animal can define a walk-method for *all* sub classes

# Class Inheritance

## Meaning

- No change in accessibility/visibility rules
  - private fields/methods still not visible, protected only within the same package etc.
- Objects of sub class can execute methods defined in super class
  - E.g., the class Animal can define a walk-method for *all* sub classes
- Objects of the sub class can be assigned to variables of the super class
  - `Animal someAnimal = new Horse();`
  - `Animal[] zooAnimals = new Animal[2] { new Horse(), new Donkey() };`
- Casting from sub class to super class ("upwards") always works
  - `Animal someAnimal = (Animal) myHorse;`

# demo

Animal and Hippo

# Inheritance

## Method Overriding

```
1  class Animal {
2    public void step(int size) { /*...*/ };
3  }
4
5  class Horse extends Animal {
6  }
7
8  class Main {
9    public static void main(String[] args) {
10     Horse h = new Horse();
11     h.step(5);
12   }
13 }
```

# Inheritance

## Method Overriding

```
1  class Animal {
2    public void step(int size) { /*...*/ };
3  }
4
5  class Horse extends Animal {
6  }
7
8  class Main {
9    public static void main(String[] args) {
10     Horse h = new Horse();
11     h.step(5);
12   }
13 }
```

- Objects of the sub class can call methods defined in super class

# Inheritance

## Method Overriding

```
1  class Animal {
2    public void step(int size) { /*...*/ };
3  }
4
5  class Horse extends Animal {
6    public void step(int size) { /*...*/ };
7  }
8
9  class Main {
10   public static void main(String[] args) {
11     Horse h = new Horse();
12     h.step(5);
13   }
14 }
```

# Inheritance

## Method Overriding

```
1  class Animal {
2    public void step(int size) { /*...*/ };
3  }
4
5  class Horse extends Animal {
6    public void step(int size) { /*...*/ };
7  }
8
9  class Main {
10   public static void main(String[] args) {
11     Horse h = new Horse();
12     h.step(5);
13   }
14 }
```

- Methods in sub class override methods in super class
- Calling super method explicitly
  - Outside of sub class by casting: `((Animal)h).step(5);`
  - Inside of sub class with `super`: `super.step(5);`
    - Think of super as `((Animal) this)` (in this case)

# Variable Type != Object Type

- Each variable has a type
  - E.g., `int`, `String`, `Horse`, ...
- Each object and value has a type
  - E.g., `int`, `String`, `Horse`, ...

# Variable Type != Object Type

- Each variable has a type
  - E.g., `int`, `String`, `Horse`, ...
- Each object and value has a type
  - E.g., `int`, `String`, `Horse`, ...
- If object/value type and variable type match, we can make an assignment
  - E.g., `int i = 5;`
  - E.g., `Horse h = new Horse();`

# Variable Type != Object Type

- Each variable has a type
  - E.g., `int`, `String`, `Horse`, ...
- Each object and value has a type
  - E.g., `int`, `String`, `Horse`, ...
- If object/value type and variable type match, we can make an assignment
  - E.g., `int i = 5;`
  - E.g., `Horse h = new Horse();`
- It's a compile error, if they do not match
  - E.g., `int i = true;` ⚠
  - E.g., `Horse h = new Donkey();` ⚠

# Variable Type != Object Type

- Each variable has a type
  - E.g., `int`, `String`, `Horse`, ...
- Each object and value has a type
  - E.g., `int`, `String`, `Horse`, ...
- If object/value type and variable type match, we can make an assignment
  - E.g., `int i = 5;`
  - E.g., `Horse h = new Horse();`
- It's a compile error, if they do not match
  - E.g., `int i = true;` ⚠
  - E.g., `Horse h = new Donkey();` ⚠
- But we can assign a object of a sub class to a variable of a super class
  - E.g., `Animal a = new Horse(); //if Horse extends Animal`

## `java.lang.Object`

- All classes inherit automatically from `java.lang.Object`
  - I.e., every object is in an instance of `java.lang.Object` (though maybe indirectly)
- Class provides a few methods

Javadoc

  - `Object clone()`
  - `boolean equals(Object obj)`
  - `int hashCode()`
  - `String toString()`
  - `void wait()`, `void wait(long timeout)`, `void wait(long timeout, int nanos)`
  - `void notify()`, `void notifyAll()`
  - `void finalize()`
  - `Class<?> getClass()`

# Testing Inheritance

- New operator: `isinstance`

```
1 Horse h = new Horse();
2
3 h instanceof Horse;  // true
4 h instanceof Object; // true
5 h instanceof String; // false
6 h instanceof Animal; // true if Horse extends Animal
```

# Remarks on Inheritance

- Why inheritance?
  - Model commonalities in our domain
  - The same behaviour can be implement as high as possible in the hierarchy, and only once
  - Again, reducing complexity

# Remarks on Inheritance

- Why inheritance?
  - Model commonalities in our domain
  - The same behaviour can be implement as high as possible in the hierarchy, and only once
  - Again, reducing complexity
- Multiple inheritance: Can a class inherit from multiple classes?
  - In Java: No
    - Because method calls then become ambiguous

# Remarks on Inheritance

- Why inheritance?
  - Model commonalities in our domain
  - The same behaviour can be implement as high as possible in the hierarchy, and only once
  - Again, reducing complexity
- Multiple inheritance: Can a class inherit from multiple classes?
  - In Java: No
    - Because method calls then become ambiguous
  - In C++/Python: Yes!
    - C++: Programmer has to resolve ambiguity with additional syntax
    - Python: Depends on the order in which inheritance has been specified

# demo

Exercise


Schöne Feiertage und Gute Erholung!