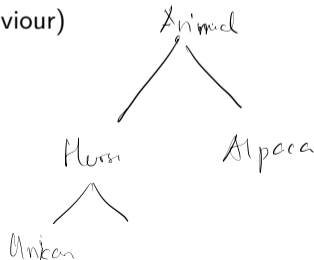
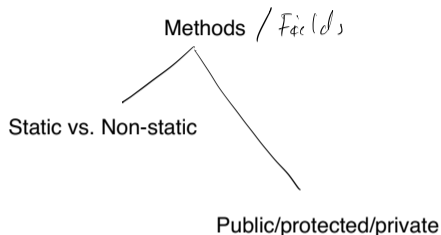


Recap

- ▶ 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



demo

Exercise 9

Understanding the class AsciiImage

Two-Dimensional Arrays

1D Array

char[] arr = new char[] { 'a', 'b', 'c', 'd' }

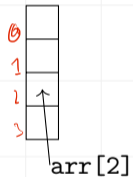
| |
|---|
| a |
| b |
| c |
| d |

Understanding the class AsciiImage

Two-Dimensional Arrays

1D Array

```
char[] arr = ...
```

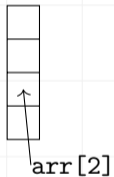


Understanding the class AsciiImage

Two-Dimensional Arrays

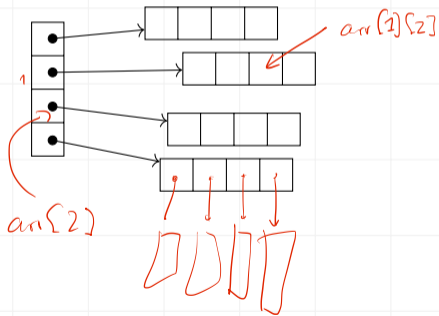
1D Array

`char[] arr = ...`



2D Array

`char[][] arr = ...`

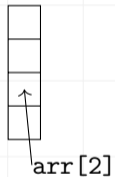


Understanding the class AsciiImage

Two-Dimensional Arrays

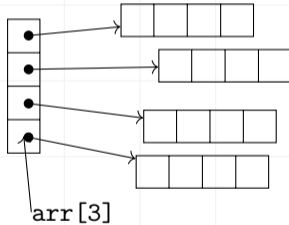
1D Array

```
char[] arr = ...
```



2D Array

```
char[][] arr = ...
```

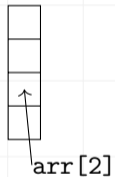


Understanding the class AsciiImage

Two-Dimensional Arrays

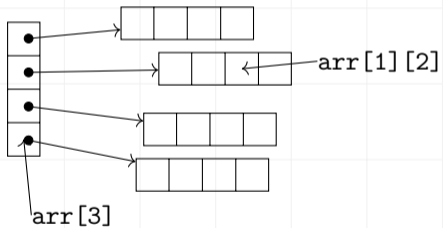
1D Array

```
char[] arr = ...
```



2D Array

```
char[][] arr = ...
```





UNIVERSITÄT
ZU KÖLN

Session 10: Methods and Inheritance

Softwaretechnologie: Java I

Nils Reiter

`nils.reiter@uni-koeln.de`

December 18, 2024



INSTITUT FÜR
DIGITAL HUMANITIES
UNIVERSITÄT ZU KÖLN

Section 1

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

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 call 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 call 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;
```

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 \neq 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 \neq 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 \neq 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 \neq 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
 - ▶ `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()`

Javadoc

Testing Inheritance

- ▶ New operator: `instanceof`

```
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 implemented 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 implemented 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

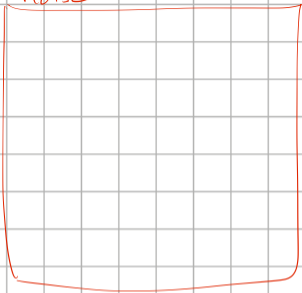
- ▶ Why inheritance?
 - ▶ Model commonalities in our domain
 - ▶ The same behaviour can be implemented as high as possible in the hierarchy, and only once
 - ▶ Again, reducing complexity
- ▶ Multiple inheritance: Can a class inherit from multiple classes?
 - ▶ In Java: No
 - ▶ Because method calls then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

Any questions about anything from the last ten weeks?

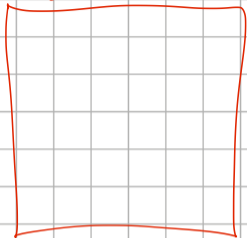
demo

Exercise

Horse



Horse



```
Horse h = new Horse();
```

```
Horse h2 = new Horse();
```

```
h.birthday();
```

```
public Horse {
```

```
    public void birthday() {
```

```
        this.age++;
```

```
    }
```

```
}
```