

Recap: Inheritance

- ▶ A class can inherit from another class
- ▶ New keyword: `extends`, used in the class declaration:

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

- ▶ Horse: sub class
- ▶ Animal: super class
- ▶ Sub class can be assigned to variables of the super type

```
1 Animal[] animals = new Animal[3];  
2 animals[0] = new Horse();
```

Section 1

Exercise 9



Session 10: Abstract Classes, Abstract Methods and Interfaces

Softwaretechnologie: Java 1

Nils Reiter

`nils.reiter@uni-koeln.de`

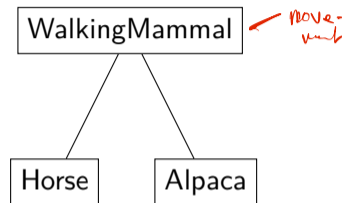
January 10, 2024

Section 2

Abstract Classes

Introduction

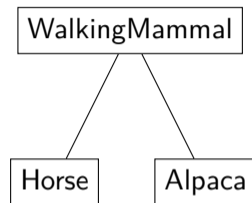
- ▶ So far: Alpaca and Horse inherit from WalkingMammal
 - ▶ Animal movement implemented in class WalkingMammal



Introduction

- ▶ So far: Alpaca and Horse inherit from WalkingMammal
 - ▶ Animal movement implemented in class WalkingMammal
- ▶ What is the problem with the statement below?

```
1 WalkingMammal wm = new WalkingMammal();
```

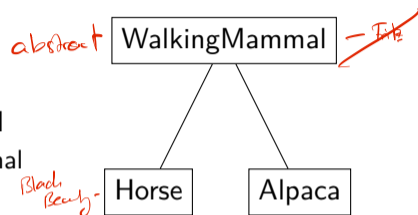


Introduction

- ▶ So far: Alpaca and Horse inherit from WalkingMammal
 - ▶ Animal movement implemented in class WalkingMammal
- ▶ What is the problem with the statement below?

```
1 WalkingMammal wm = new WalkingMammal();
```

- ▶ We often introduce super classes for good reasons
- ▶ But creating an instance of them doesn't make sense
- ▶ By declaring a class as `abstract`, we can prevent its instantiation



Abstract Classes

Example

```
1 public abstract class WalkingMammal {
2     public void walk() {
3         // ...
4     }
5 }

1 public class Horse extends WalkingAnimal {
2     // ...
3 }

1 public class Main {
2     public static void main(String[] args) {
3         WalkingMammal wm = new WalkingMammal(); // compile error
4         Horse h = new Horse(); // works as before
5     }
6 }
```


Abstract Classes

```
WM[] animals = new WM[5];  
animals[0] = new Horse();
```

- ▶ An abstract class is a regular class
- ▶ Only exception: It's impossible to instantiate it
- ▶ Use case: Inheriting from it, and create an instance of the sub class

Section 3

Abstract Methods

Introduction

- ▶ All animals are capable of reproduction
- ▶ But their methods are very different:
 - ▶ Mammals generally give birth to live young
 - ▶ Except the platypus, which lays eggs
 - ▶ Lizards generally lay eggs
 - ▶ Except the common lizard, which may also have live young




Introduction

- ▶ All animals are capable of reproduction
- ▶ But their methods are very different:
 - ▶ Mammals generally give birth to live young
 - ▶ Except the platypus, which lays eggs
 - ▶ Lizards generally lay eggs
 - ▶ Except the common lizard, which may also have live young
- ▶ The only sensible place for an implementation of that is the specific class of an animal
- ▶ But we may want to encode that all animals are capable of reproduction somehow

Mammal
 |
 WH
 |
 House - mate()



Introduction

- ▶ All animals are capable of reproduction
- ▶ But their methods are very different:
 - ▶ Mammals generally give birth to live young
 - ▶ Except the platypus, which lays eggs
 - ▶ Lizards generally lay eggs
 - ▶ Except the common lizard, which may also have live young
- ▶ The only sensible place for an implementation of that is the specific class of an animal
- ▶ But we may want to encode that all animals are capable of reproduction somehow
- ▶ `abstract` methods allow us to do this
- ▶  `abstract` means something else for classes than for methods



Abstract Methods

Exmample

```
1 public abstract class Animal { note
2   public abstract Animal reproduce(Animal other);
3 }
```

```
1 public class Horse extends Animal {
2   public Animal reproduce(Animal other) {
3     // ...
4   }
5 }
```

```
1 public class Main {
2   public static void main(String[] args) {
3     Horse h1 = new Horse();
4     Horse h2 = new Horse();
5     Horse h3 = h1.reproduce(h2);
6   }
7 }
```

Abstract Methods and Classes

- ▶ If a class has one abstract method, the class must be abstract as well
 - ▶ Because otherwise, there could be an object with a method that doesn't have an implementation

Abstract Methods and Classes

- ▶ If a class has one abstract method, the class must be abstract as well
 - ▶ Because otherwise, there could be an object with a method that doesn't have an implementation
- ▶ A non-abstract class that inherits from an abstract class, must implement all abstract methods

demo

Method Signature and Overriding

- ▶ Method signature: Name, argument types, argument order
 - ▶ Return type not part of the signature
- ▶ Overriding a method in a subclass
 - ▶ Provide a method with the same signature and “matching return type”

Method Signature and Overriding

- ▶ Method signature: Name, argument types, argument order
 - ▶ Return type not part of the signature
- ▶ Overriding a method in a subclass
 - ▶ Provide a method with the same signature and “matching return type”
- ▶ Matching return type
 - ▶ The overriding method (i.e., the one in the sub class) can return a more specific type
 - ▶ Because the more specific type is also of the more generic type

Method Signature and Overriding

- ▶ Method signature: Name, argument types, argument order
 - ▶ Return type not part of the signature
- ▶ Overriding a method in a subclass
 - ▶ Provide a method with the same signature and “matching return type”
- ▶ Matching return type
 - ▶ The overriding method (i.e., the one in the sub class) can return a more specific type
 - ▶ Because the more specific type is also of the more generic type

Example

```
1 Animal a = h1.reproduce(h2); // because the return type of reproduce,  
2 // as defined in Horse is more specific than as defined in Animal
```

Section 4

Interfaces

Introduction

- ▶ An `interface` is similar to a class, in which
 - ▶ All methods are abstract
 - ▶ All methods are public
 - ▶ There are no fields

demo

Section 5

Exercise