Session 8: Static, Private, Public, Protected

Softwaretechnologie: Java I

Nils Reiter
nils.reiter@uni-koeln.de

November 30, 2022

INSTITUT FÜR
DIGITAL HUMANITIES
UNIVERSITÄT ZU KÖLN

Section 1

Exercise 7

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

Recap

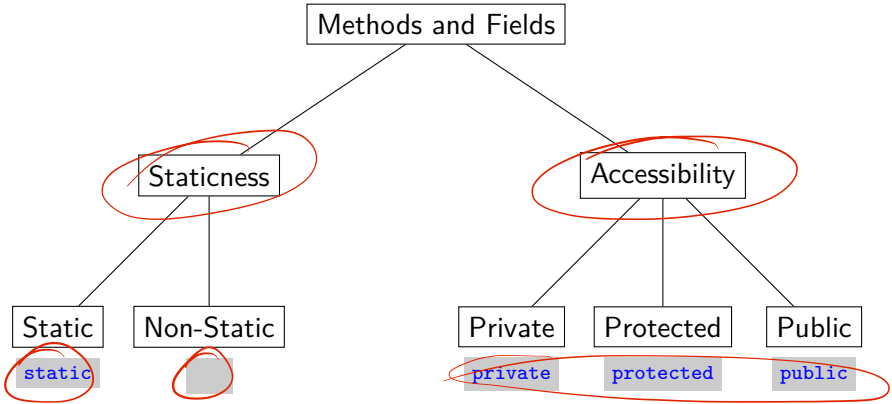
Example

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

  // 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();
    // create a second instance of type horse
    Horse h2 = new Horse();
  }
}
```

Section 2

Methods

Introduction



Staticness

horse1.mate(..)

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

Section 3

Exercise