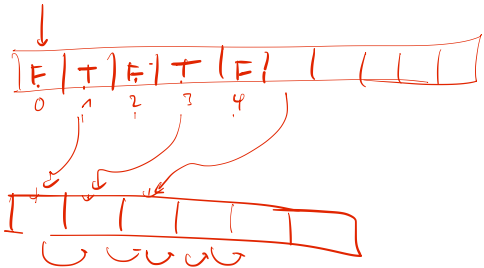
Session 7: Classes and Objects

Softwaretechnologie: Java I

Nils Reiter
nils.reiter@uni-koeln.de

November 23, 2022

INSTITUT FÜR
DIGITAL HUMANITIES
UNIVERSITÄT ZU KÖLN


F
T
F
T
F
0
1
2
3
4

Section 1

Exercise 6

Section 2

Object-Oriented Programming

Introduction

- Paradigm on how to write and structure code
- Method for dealing with complexity
- Very popular across many programming languages
- Classes and objects to structure our domain of interest

Classes and Instances

Classes

- A unit of data and behaviour
- Represents all things of the same type in our domain
- Data: Stored in fields
- Behaviour: Defined in methods

Instances (= objects)

- An instance of a class C is one individual of the type
- All instances of C have the same fields, but (potentially) with different values
- Their class determines what they can do

Classes and Instances

Example

- Horses
  - Can run fast
  - Give birth to live young (= are mammals)
  - Can be grey, brown, white, ...
  - Life span: 25–30 years
- Cranes
  - Can fly
  - Lay eggs
  - Are grey with a black-ish neck
  - Life span: 20–30 years

Classes in Java

```
1  public class Horse {
2    // the fields/variables of a class to store data about an instance
3    String color;
4    String name;
5    int currentSpeed;
6
7    // methods of the class to define their behaviour
8    public Horse mate(Horse partner) {
9      // two horses meet and make a new horse
10   }
11
12   public static void main(String[] args) {
13     // create an instance of type horse
14     Horse h1 = new Horse();
15     // create a second instance of type horse
16     Horse h2 = new Horse();
17   }
18 }
```

Fields

Methode

demo

Classes in Java

- Class definitions are introduced with `class`
  - A class name is used instead of a type
  - I.e., we can define our own types
- Classes can have fields to store values and methods to do something
  - Methods work like functions, except that they are not `static` anymore
  - But they have a regular return value
- Each (public) class is in their own file

Object Initialisation

```
1  public class Horse {
2    // newly created horses have zero age
3    int age = 0;
4
5    // Constructor: Special function called when an object is created
6    // Doesn't have a return type, otherwise a normal function
7    // with the same name as the class
8    public Horse() {
9      System.out.println("A horse is born.");
10   }
11
12   public static void main(String[] args) {
13     Horse h1 = new Horse(); // "A horse is born" gets printed
14   }
15 }
```

Object Initialisation

Constructor

- A regular function, but
  - Without return type
  - With the same name as the class

Object Initialisation

Constructor

- A regular function, but
  - Without return type
  - With the same name as the class
- Can take arguments:

```
1 public class Horse {
2   String theName;
3
4   public Horse(String name) {
5     theName = name;
6   }
7
8   public static void main(String[] args) {
9     Horse h1 = new Horse("Joe");
10  }
11 }
```

The Keyword `this`

- `this` is a special variable
- Within a method, `this` refers to the object used to call the method

```
1  public class Horse {
2    String name;
3
4    public Horse(String name) { this.name = name; }
5
6    public void printName() { System.out.println(this.name); }
7
8    public static void main(String[] args) {
9      Horse h1 = new Horse("Joe");
10     Horse h2 = new Horse("Mary");
11     h1.printName(); // prints Joe
12     h2.printName(); // prints Mary
13   }
14 }
```

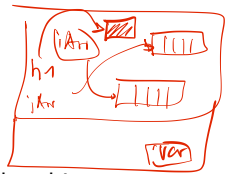
Reference Types and `null`

- All classes are reference types
  - When dealing with a variable, we're dealing with a reference to the object
  - If two objects are created with `new`, they are not equal (`==`), even if they have the same field values

Reference Types and `null`

- All classes are reference types
  - When dealing with a variable, we're dealing with a reference to the object
  - If two objects are created with `new`, they are not equal (`==`), even if they have the same field values
- In some situations, we need to signify that a reference is empty
  - I.e., it's a variable of a certain reference type, but an object is not yet created
  - E.g., if two horses mate, but don't produce an offspring

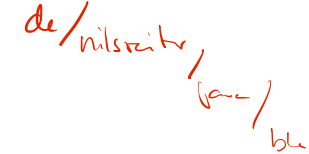
Reference Types and `null`



- All classes are reference types
  - When dealing with a variable, we're dealing with a reference to the object
  - If two objects are created with `new`, they are not equal (`==`), even if they have the same field values
- In some situations, we need to signify that a reference is empty
  - I.e., it's a variable of a certain reference type, but an object is not yet created
  - E.g., if two horses mate, but don't produce an offspring
- A new keyword: `null`
  - `null` is a value for any reference type
  - E.g., `Horse h = null;` established such an empty reference

Packages

- Multiple classes often belong conceptually together
- Packages can be used to group classes (and files)
- Package declaration: `package de.nilsreiter.java.bla;`
  - First statement within a file
  - Package hierarchy must reflect directory hierarchy
    - Eclipse hides that from us
- Package name conventions
  - Lower-cased
  - Reversed URLs to be globally unique



Section 3

Exercise