Recap: Strings

- Strings: Sequences of characters to represent texts
  - String is a class
  - Many methods are implemented to help dealing with Strings

demo

Exercise 7

Programmieren mit ChatGPT, Co-Pilot & co.

- Programmieren mit LLM-Unterstützung ist gerade trendy
- Sprachmodelle generieren oft syntaktisch korrekten Programmcode
- Offene Fragen
  - Sicherheit des erzeugten Codes — Fu et al. (2023)
  - Urheberrecht und Geheimnisschutz, Effizienz — Alizadeh et al. (2024)
  - Software-Abhängigkeiten nach außen — Corso et al. (2024)
  - Geschäftsmodell von KI-Anbietern

Programmieren mit ChatGPT, Co-Pilot & co.

- Programmieren mit LLM-Unterstützung ist gerade trendy
- Sprachmodelle generieren oft syntaktisch korrekten Programmcode
- Offene Fragen
  - Sicherheit des erzeugten Codes — Fu et al. (2023)
  - Urheberrecht und Geheimnisschutz, Effizienz — Alizadeh et al. (2024)
  - Software-Abhängigkeiten nach außen — Corso et al. (2024)
  - Geschäftsmodell von KI-Anbietern

Was heißt das für Sie?

- Man lernt nur, indem man sich mit etwas (selbst) beschäftigt
- Sprachmodelle machen Fehler, die Sie (Stand jetzt) nicht erkennen (können)
  - ⚠ Das kann sich auch auf die Erklärungen beziehen, die generiert werden
- Wenn Sie lernen (nur) mit LLM-Unterstützung zu programmieren, machen Sie sich abhängig und sind (später) weniger flexibel

UNIVERSITÄT
ZU KÖLN

Session 8: Classes and Objects

Softwaretechnologie: Java I

Nils Reiter
nils.reiter@uni-koeln.de

December 4, 2024

INSTITUT FÜR
DIGITAL HUMANITIES
UNIVERSITÄT ZU KÖLN

Introduction

- Object-Oriented Programming (OOP): Paradigm on how to write and structure code
- Method for dealing with complexity – not technically necessary
- Very popular across many programming languages
- Classes used to structure our domain of interest

Structuring our Domain of Interest

- Domain: What is our program about?
  - The “world” of our program
- A number of things
- But things belong to certain categories

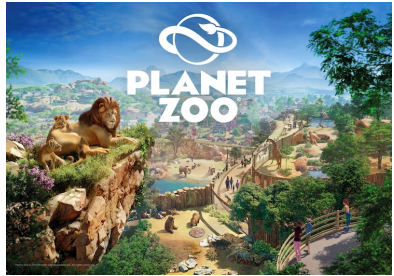
Domains

Beispiel: Universität

- Idee: Nächste Version von Klips oder neue Erweiterung von Sims
- Dinge: Philosophikum, Fahrradgarage, dieser Programmierkurs, Elisa Cugliana, Øyvind Eide, Stefan Schneider (ausgedachter Studentenname), Azra Özgür (ausgedachter Studentinnenname), der Studiengang Informationsverarbeitung...
- Kategorien: Gebäude, Kurse, Dozent:innen, Student:innen, Prüfungen, Studiengänge, ...

Domains

Beispiel: Zoo



- Dinge: Schuppen, Heinz, Hexe, Pommesbude, Heinz (Arzt), ...
- Kategorien: Krankheit, Giraffe, Nahrung, Fressbude, Zoowärter, ...
  - Säugetier
    - Tier

KÖLNER ZOO

Classes and Instances

Classes / categories

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

Instances / objects

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

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

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

  // Horse starts to accelerate
  public void accelerate(int rate) {
    currentSpeed = currentSpeed + rate;
  }

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

demo

Horse

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

```java
public class Horse {
  // newly created horses have zero age
  int age = 0;

  // Constructor: Special function called when an object is created
  // Doesn't have a return type, otherwise a normal function
  // with the same name as the class
  public Horse() {
    System.out.println("A horse is born.");
  }

  public static void main(String[] args) {
    Horse h1 = new Horse(); // "A horse is born" gets printed
  }
}
```

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 }
```

Object Lifecycle

Three usage scenarios for an object

- Object creation
  - `new Horse()` calls the constructor
- Object use (`myHorse` is an object of type `Horse`)
  - `myHorse.run()` calls the method run
  - `myHorse.name` accesses a field of the object
- Object deletion
  - Done automatically through garbage collection
  - ⚠ This is different in C++

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

demo

Exercise 8

References I

- Alizadeh, Negar/Boris Belchev/Nishant Saurabh/Patricia Kelbert/Fernando Castor (2024). *Analyzing the Energy and Accuracy of LLMs in Software Development*. DOI: 10.48550/arXiv.2412.00329. arXiv: 2412.00329[cs]. URL: http://arxiv.org/abs/2412.00329 (visited on 12/04/2024).
- Corso, Vincenzo/Leonardo Mariani/Daniela Micucci/Oliviero Riganelli (2024). "Generating Java Methods: An Empirical Assessment of Four AI-Based Code Assistants". In: Publisher: arXiv Version Number: 2. DOI: 10.48550/ARXIV.2402.08431. URL: https://arxiv.org/abs/2402.08431 (visited on 12/04/2024).
- Fu, Yujia/Peng Liang/Amjed Tahir/Zengyang Li/Mojtaba Shahin/Jiaxin Yu/Jinfu Chen (2023). *Security Weaknesses of Copilot Generated Code in GitHub*. Version Number: 2. DOI: 10.48550/ARXIV.2310.02059. URL: https://arxiv.org/abs/2310.02059 (visited on 12/04/2024).