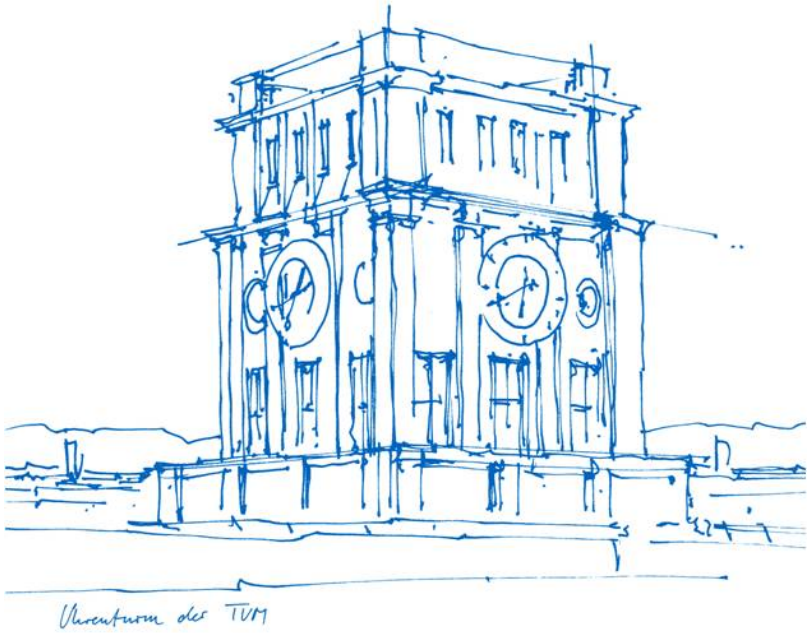
Grundlagenpraktikum: Programmierung Einführung

Technische Universität München

TUM School of Computation, Information and Technology

PGDP TUT-Fr-08-a (Zulip)
MI 00.13.008
01.11.2024
Shiqi Li



Schön dass ihr hier seid!

- Ist das Tutorium zu voll?
  - gerne wechseln, einige freie Plätze in den anderen Tutorien.
- Ist das Tutorium eher ruhig?
  - Genießt die individuelle Betreuung.
  - Kann leider sein, dass wir Termine absagen oder zusammenlegen wenn es zu leer wird.

Ablauf P-Aufgaben

- Es kann sein dass wir nicht alle schaffen, es ist OK
- Ihr sollt die Aufgaben selber machen
- Stellt gerne Fragen, dafür sind wir da

Hinweise zu H-Aufgaben (1)

- Das ist recht viel Text
  - Absicht! Die Forderungen zu verstehen gehört dazu, hier wie im Alltag einer Informatikerin
  - ÜPA Aufgaben sind kürzer

Hinweise zu H-Aufgaben (2)

- Das ist recht viel zu tun
  - Programmieren ist ein Handwerk
  - Programmieren ist schwierig
  - Man lernt durch viel Übung

Zulip

- Stellt gerne Fragen (aber erst suchen!)
- Versucht minimale Beispiele zu geben
- Ihr werdet Gegenfragen bekommen, z.B:
  - Was hast du schon probiert?
  - Was hast du erwartet?
  - Was ist passiert?
- Das ist keine Kritik! Das sind wichtige Informationen für uns
- Also am liebsten gleich mitgeben
- Das hilft euch auch, das Problem zu verstehen

Data Types

- Variable im Programm hat ein Data Type
- Einfaches Struktur einer Variable: *type name = genauer Inhalt;*

byte	1 Byte (8 bits)	Ganzzahl : -128 ~ 127
short	2 Bytes (16 bits)	Ganzzahl : -2^16~ 2^16-1
int	4 Bytes (32 bits)	Ganzzahl : -2^32~ 2^32-1
long	8 Bytes (64 bits)	Ganzzahl : -2^64~ 2^64-1

```
int a; // Deklaration a variable with type int
a = 1; // initialization
a = 2; // modify
```

=>Vereinfachen

```
// Deklaration and initialization
int a = 1;
a = 2; // modify
```

Data Types

float	4 Bytes (32 bits)	Initialization mit ‚f' Suffix
double	8 Bytes (64 bits)	
boolean	1 bit	true or false
char	2 bits	character or ASCII values
String*		Text

*: String ist nicht als Data Type sondern eine Klasse in Java, die eigene Methoden hat

Beispiele:

```
float f = 1.23f;
double d = 4.56;
boolean b = true;
char c = 'a';
char c_1 = 97;// output character a; in ASCii table 97 is a
String s = "Hello Java!";
```

Objekt oriented Programming (OOP) - Java

- zwei Hauptaspekte von OOP : **Klasse** und **Objekt**
- Klasse ist eine Vorlage für Objekte, Objekt ist eine **Instanz** einer Klasse.
  - Objekt hat **alle** Attribute seiner Klasse und
  - Objekt ausführt Aktionen durch Methoden seiner Klasse.
- Eine Klasse kann **mehr** Objekte instanziieren.

OOP - Klasse

Klasse besteht aus:

- Attributen (Member Variable)
- Konstruktor
- Methoden (Member Methode)
- Getter and Setter

* Folgende Beispiele: aus W02P01, eine Klasse Dog als Beispiele

  - Eigenschaften von Dog: age, weight, gender, race, color, usw.
  - Aktion von Dog: play, eat, drink als Beispiele

OOP: Klasse - Attribute

- Attribute — i.allg. Eigenschaften von Klasse
- alle Objekte haben, aber Variablen unterscheiden sich voneinander
  - Konstrukt : Modifier + (override) + type + name: *private int age;*
    - Modifier (zukünftige Woche): im UML : + für public ; - für privat
    - override (zukünftige Woche): (leer) : override erlaubt final: override nicht erlaubt
    - Beispiele : Ann und Bee sind zwei Objekte einer Klasse „Dog“

Objekt Ann von Dog Klasse
Age : 5
Weight: 25.3 (kg)
Gender: w
Race: Golden Retriever
Color: golden

Objekt Bee von Dog Klasse
Age : 3
Weight: 14.0 (kg)
Gender: m
Race: Corgi
Color: golden and white

OOP: Klasse-Methode

- Methoden — Aktionen : Aktion ausführen
  - alle Objekte machen können, dasselbe Aktion mit verschiede Parameter
  - Konstrukt: Methode head und Methode body.
    - Methode heat: Verkapselung, return value type, Methode Name, Parameter
    - Return value: Nach dem Aufruf einer Methode gibt es eine Value als Ergebnis von Aktion. **Void** für **keine** Value gegeben
    - Parameter: Variable aus die aufruft Methode, nützlich bei der Ausführung von Aktion
    - Methode body: Anweisungen stellen, zur Ausführung von Aktionen
  - Beispiel

```
Verkapselung  returned Value Type  Methode Name    Parameter
 public       void                 essen           (String meat) {
     // Methode body
 }
```

OPP: Klasse - Konstruktor

- Konstruktor: spezielle Methode von Klasse zum Initialisieren von einem Objekt
  - public Methode und ohne return Value Type, Klassenname als Methodenname
  - Parameter zum Initialisieren von der Attribute eines Objekts
  - Nicht im UML Diagramm
  - Sonderfall: Name von Konstruktor Anfangen mit Großbuchstabe

OPP: Klasse - Konstruktor

- Beispiel :

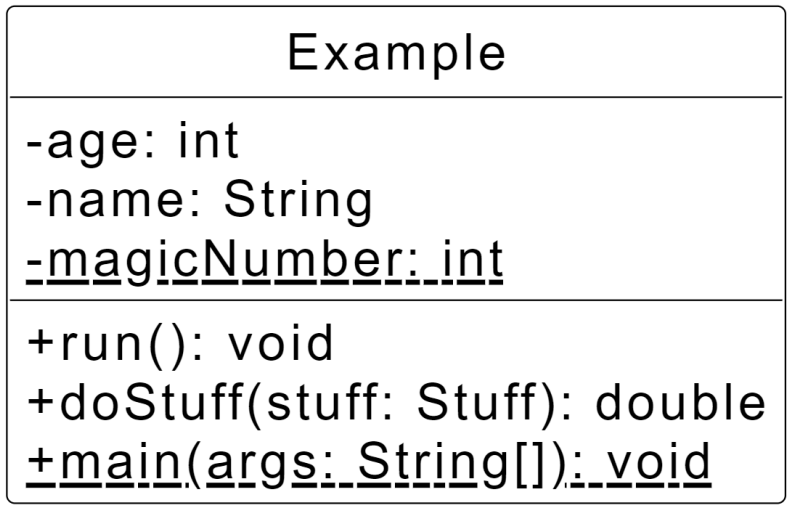
```java
public class Dog {  new *
    private int age;  1 usage
    private double weight;  1 usage
    private String gender;  1 usage
    private String race;  1 usage
    private String color;  1 usage

    public Dog(int age, double weight, String gender, String race, String color) {
        this.age = age; // "=" bedeutet nicht Vergleich sondern Abtretung
        this.weight = weight; // variable mit this keyword: Member Variable
        this.gender = gender;
        this.race = race;
        this.color = color;
    }
    public static void main(String[] args) {  new *
        int age = 5;
        double weight = 25.3;
        String gender = "w";
        String race = "Golden Retriever";
        String color = "golden";
        // Object initialisieren
        Dog ann = new Dog(age, weight, gender, race, color);
    }
```

OOP: Klasse - Getter & Setter

- access Modifier beider Methoden: public
- externe Klassen durch Getter und Setter die private Attribute einer Klasse zugreifen.
- Get Methode: Return Variable von privater Attribute
- Set Methode: Modify Variable von privater Attribute
- Beispiel

```
public double getWeight() {  no usages  new *
    return weight;
}
public void setWeight(double weight) {  no usages  new *
    this.weight = weight;
}
```

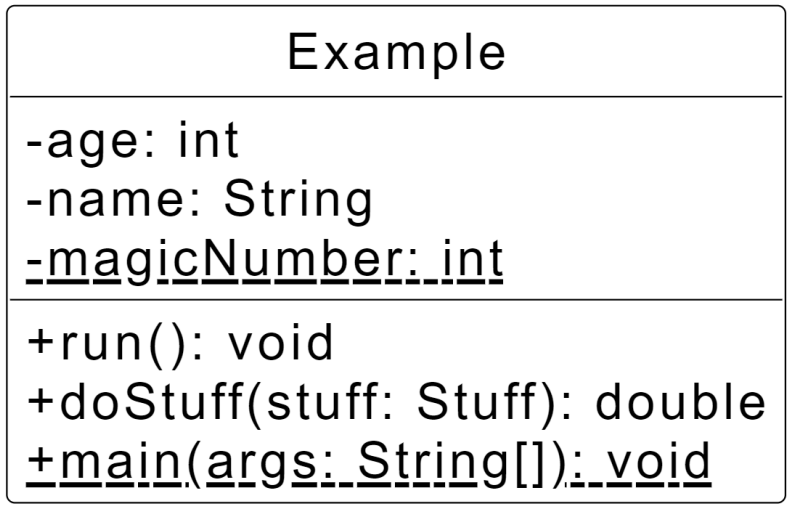
UML



- Oben sind Attribute, unten sind Methoden
- Konstruktoren, Getter und toString sind üblicherweise nicht im Diagramm
- + public, jeder kann zugreifen
- - private, nur die eigene Klasse kann zugreifen
- static, Klassenattribut / -methode

  statische Variable kann zugegriffen werden
  Ohne Initialisieren Objekt einer Klasse
  (zukünftigen Woche)

UML zu Code




```java
public class Example {
  private int age;
  private String name;

  private static int magicNumber;

  public void run() {}

  public double doStuff(Stuff stuff) {
    return 0.0;
  }

  public static void main(String[] args) {}
}
```

W02P01 - Verhaltensforschung

Bearbeite nun die Aufgabe W02P01 - Verhaltensforschung

W02P02 - Fahrzeuge

Bearbeite nun die Aufgabe W02P02 - Fahrzeuge

W02P03 - Roboter I

Bearbeite nun die Aufgabe W02P03 - Roboter I

W02P04 - Roboter II

Bearbeite nun die Aufgabe W02P04 - Roboter II

Slide und Code von Aufgaben

GitHub : Code aller P-Aufgaben

- https://github.com/SikiaLi/PGDP.git