Karlsruher Institut für Technologie

Institut für Technik der Informationsverarbeitung

Institutsleitung
Prof. Dr.-Ing. Dr. h.c. J. Becker
Prof. Dr.-Ing. E. Sax
Prof. Dr. rer. nat. W. Stork

Übung 05: Informationstechnik (IT)

Daniel Grimm

Teil 1: Besprechung Aufgabenblatt 5

Aufg. 5.01: Verständnisfragen Lsg. (1)

a) Um den kontrollierten Zugriff auf die Datenelemente einer Klasse sicherzustellen, werden die Datenelemente normalerweise als `private` deklariert und durch `get` & `set`-Methoden gelesen oder geändert.

b) Innerhalb einer Methode der Klasse können folgende Elemente dieser Klasse direkt mit ihrem Namen angesprochen werden:

i. alle Elemente Richtig

ii. nur die Datenelemente Falsch

iii. nur die public-Elemente Falsch

c) Um beim Aufruf einer Methode den für den Hin- und Rücksprung notwendigen Overhead zu vermeiden, kann eine Methode als `inline` definiert werden.

Aufg. 5.01: Verständnisfragen Lsg. (2)

d) Wenn zwei verschiedene Objekte derselben Klasse angelegt werden, wird der Maschinencode für jede Methode auch zweimal erzeugt.
→ Falsch

e) Bei einem Funktionsaufruf kann der durch das Erzeugen und Zerstören von Objekten bedingte Overhead vermieden werden, wenn Argumente per Referenz oder Zeiger übergeben werden.

Aufg. 5.01: Verständnisfragen Lsg. (1)

f) Die Methoden einer abgeleiteten Klasse können auf die **`private`**-Elemente der Basisklasse zugreifen. → Falsch

g) Eine abgeleitete Klasse erbt eine **`public`**-Methode der Basisklasse nicht, wenn sie selbst eine **`public`**-Methode mit gleichem Namen besitzt. → Falsch (unterschiedlicher Prototyp oder bei gleichem Prototyp über Basisklassenoperation)

h) Methoden einer abgeleiteten Klasse können auf **`protected`** Elemente der Basisklasse zugreifen. Die **`protected`** Elemente sind aber von außerhalb der Basisklasse und deren Kindklassen nicht zugreifbar. → Richtig

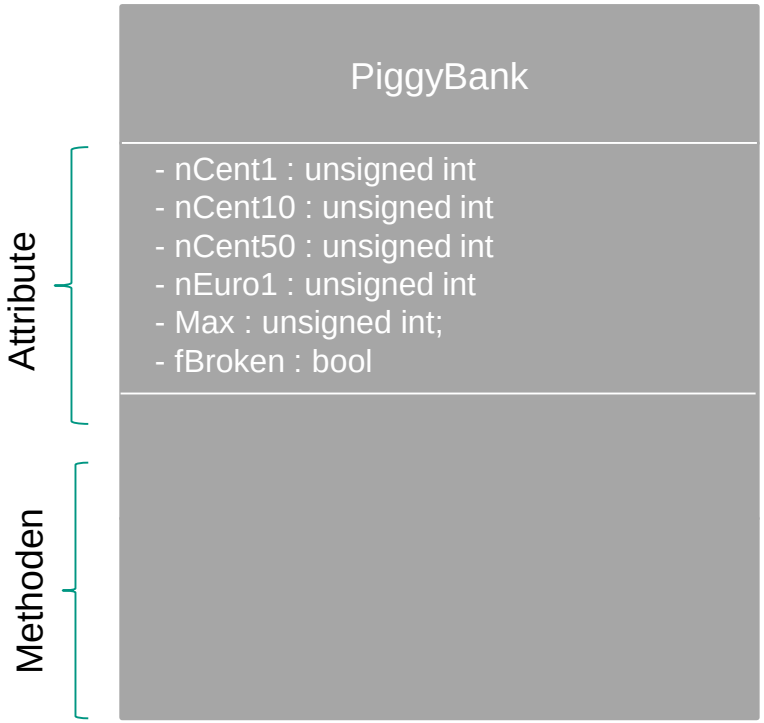
- Aufg. 5.01: Verständnisfragen

Aufg. 5.02: OOP, Definition von Klassen

- Definieren Sie eine Klasse PiggyBank zur Verwaltung der Münzen in einem Sparschwein.
- Die Klasse besitzt dabei private und öffentliche Attribute und Methoden.
- Stellen Sie die Klasse PiggyBank in einem Klassendiagramm dar.
- Geben Sie die Definition der Klasse PiggyBank sowie die Definitionen der inline Methoden in der Header-Datei "PiggyBank.h" an. In einer separaten Quelldatei "PiggyBank.cpp" sollen die restlichen Methoden implementiert werden.
- Testen Sie die Klasse PiggyBank mit einem Anwendungsprogramm in einer separaten Quelldatei.

Aufg. 5.02: OOP, Definition von Klassen

- Stellen Sie die Klasse PiggyBank in einem Klassendiagramm dar.



Von außen unsichtbar → private, '-'

Die Klasse besitzt folgende von außen unsichtbare Datenelemente:

Zählvariablen, Anzahlen → unsigned int

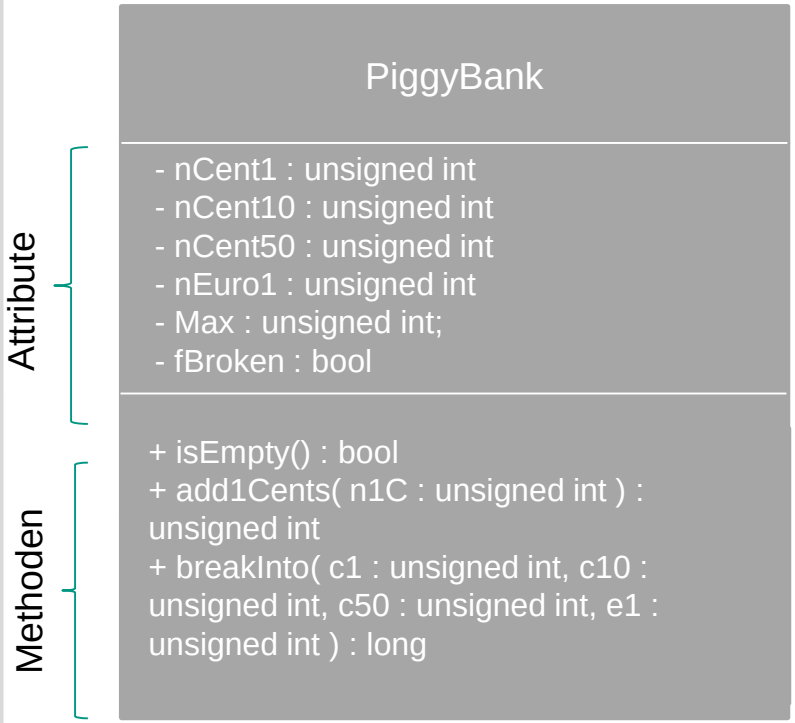
- Zählervariablen für vier Arten von Münzen (Anzahl 1-Cent, 10-Cent, 50-Cent und 1-Euro Münzen).
- Eine Variable zur Speicherung der maximalen Anzahl an Münzen, die in das Sparschwein passen.

Flag, nur zwei Zustände → bool

- Ein Flag, um anzuzeigen, dass das Sparschwein aufgebrochen wurde.

Aufg. 5.02: OOP, Definition von Klassen

- Stellen Sie die Klasse PiggyBank in einem Klassendiagramm dar.



Öffentlich → public, '+'

Des Weiteren soll die Klasse folgende öffentliche Methoden enthalten:

bool-Funktion, keine Paramter

- `isEmpty()`: liefert *true*, wenn das Sparschwein leer ist, sonst *false*. Es handelt sich hier um eine explizite inline Methode.

int-Funktion, int als Parameter

- `add1Cents()`: "wirft" eine übergebene Anzahl von 1-Cent-Münzen in das Sparschwein und liefert den Return-Wert 0, falls alle Münzen in das Sparschwein passen. Wenn das Sparschwein "überläuft", liefert die Methode die Anzahl der Münzen, die nicht mehr ins Sparschwein passen, als Return-Wert zurück.

int-Funktion, int-Referenzen als Parameter (hier nicht dargestellt)

- `breakInto()`: bricht das Sparschwein auf und liefert die gefundenen 1-Cent-, 10-Cent-, 50-Cent- und 1-Euro-Münzen mithilfe von Referenzparametern zurück. Der Zähler der Geldstücke wird auf 0 zurückgesetzt. Der Return-Wert ist der angesparte Geldbetrag in Cents.

Aufg. 5.02: OOP, Definition von Klassen

Lsg. (1)

Karlsruher Institut für Technologie

piggyBank.h

```
#ifndef _PIGGYBANK_
#define _PIGGYBANK_

class PiggyBank {
  private:
    unsigned int nCent1, nCent10, nCent50, nEuro1;
    unsigned int max;
    bool fBroken;
  public:
    PiggyBank( unsigned int m ) {
      max = m;
      nCent1 = nCent10 = nCent50 = nEuro1 = 0;
      fBroken = false;
    }
    ~PiggyBank() {}
    bool isEmpty();
    bool isFull();
    bool isBroken();
    unsigned int add1Cents( unsigned int n1C );
    unsigned int add10Cents( unsigned int n10C );
    unsigned int add50Cents( unsigned int n50C );
    unsigned int add1Euros( unsigned int n1E );
    unsigned long breakInto( unsigned int& c1, unsigned int& c10,
                             unsigned int& c50, unsigned int& e1 );
};
```

Präprozessor-Direktiven: Header-Datei wird nur einmal eingebunden

Maximale Anzahl an Münzen

Aufgebrochenes Sparschwein

Implizit inline Konstruktor

Implizit inline dummy-Destruktor

Methoden zum Überprüfen bestimmter Bedingungen anhand der privaten Attribute

Aufg. 5.02: OOP, Definition von Klassen

Lsg. (2)

piggyBank.h

```
//...

inline bool PiggyBank::isEmpty() {
  unsigned int cur = nCent1 + nCent10 + nCent50 + nEuro1;
  return cur == 0;
}


inline bool PiggyBank::isFull() {
  unsigned int cur = nCent1 + nCent10 + nCent50 + nEuro1;
  return cur >= max;
}


inline bool PiggyBank::isBroken() {
  return fBroken;
}

#endif
```

Definitionen der explizit inline Methoden der Klasse

`inline` Schlüsselwort!

Methode durch `::` Operator definieren

Ende des Präprozessor-Blocks
Ende der Header-Datei

Aufg. 5.02: OOP, Definition von Klassen

Lsg. (3)

piggyBank.cpp

```
#include "piggyBank.h"

unsigned int PiggyBank::add1Cents( unsigned int n1C ) {
  unsigned int cur = nCent1 + nCent10 + nCent50 + nEuro1;
  if( n1C <= max - cur ) {
    nCent1 += n1C;
    return 0;
  }
  else {
    nCent1 +=  ( max - cur );
    return (n1C - ( max - cur ));
  }
}

unsigned int PiggyBank::add10Cents( unsigned int n10C ) {
  unsigned int cur = nCent1 + nCent10 + nCent50 + nEuro1;
  if( n10C <= max - cur ) {
    nCent10 += n10C;
    return 0;
  }
  else {
    nCent10 += ( max - cur );
    return n10C - ( max - cur );
  }
}
```

Enthält die Definition der Klasse

Falls alle Münzen ins Sparschwein passen

Sparschwein nimmt so viele Münzen wie möglich

Anzahl restlicher Münzen

Aufg. 5.02: OOP, Definition von Klassen

Lsg. (4)

piggyBank.cpp

```
unsigned int PiggyBank::add50Cents( unsigned int n50C ) {
  // analog zu den oberen Methoden
}

unsigned int PiggyBank::add1Euros( unsigned int n1E ) {
  // analog zu den oberen Methoden
}

unsigned long PiggyBank::breakInto( unsigned int& c1, unsigned int& c10,
                                    unsigned int& c50, unsigned int& e1 ) {
  fBroken = true;
  c1 = nCent1; c10 = nCent10; c50 = nCent50; e1 = nEuro1;
  nCent1 = nCent10 = nCent50 = nEuro1 = 0;
  unsigned long sum = c1 + c10 * 10 + c50 * 50 + e1 * 100;
  return sum;
}
```

Referenz Parameter

Münzenanzahl zurückgesetzt

Rückgabewert ist der Geldbetrag in Cents

Aufg. 5.02: OOP, Definition von Klassen

Lsg. (5)

main.cpp

```
// Testprogramm
#include <iostream>
#include "piggyBank.h"

using namespace std;

void showCoins( unsigned int c1,  unsigned int c10,
                unsigned int c50, unsigned int e1 ) {
  if( c1 > 0 ) cout << c1  << " 1-Cent Stuecke"  << endl;
  if( c10 > 0 ) cout << c10 << " 10-Cent Stuecke" << endl;
  if( c50 > 0 ) cout << c50 << " 50-Cent Stuecke" << endl;
  if( e1 > 0 ) cout << e1  << " 1-Euro Stuecke"  << endl;
}

int main() {
  unsigned int c1 = 0, c10 = 0, c50 = 0, e1 = 0;
  PiggyBank myPiggy( 500 );

  if( myPiggy.isEmpty() ) {
    cout << "Das Sparschwein ist leer!" << endl;
  }

  //...
```

Enthält die Definition der Klasse

Funktion zur Ausgabe der gesammelten Münzen

Erzeugen eines Objekts der Klasse **`PiggyBank`** durch Aufruf des Konstruktors

Aufruf einer Methode der Instanz der Klasse **`PiggyBank`**

Aufg. 5.02: OOP, Definition von Klassen

Lsg. (6)

main.cpp

Solange das Sparschwein nicht voll ist

```
while( !myPiggy.isFull() ) {
  cout << endl << "Ersparnisse machen gluecklich!" << endl;
  cout << "Werfen Sie einige 1-Cent Stuecke ein: ";
  if( !( cin >> c1 ) || myPiggy.add1Cents( c1 ) != 0 ) break;

  cout << "Werfen Sie einige 10-Cent Stuecke ein: ";
  if( !( cin >> c10 ) || myPiggy.add10Cents( c10 ) != 0 ) break;

  cout << "Werfen Sie einige 50-Cent Stuecke ein: ";
  if( !( cin >> c50 ) || myPiggy.add50Cents( c50 ) != 0 ) break;

  cout << "Werfen Sie einige Euros ein: ";
  if( !( cin >> e1 ) || myPiggy.add1Euros( e1 ) != 0 ) break;
}

cout << "Vielen Dank!" << endl;

//…
```

Aufg. 5.02: OOP, Definition von Klassen

Lsg. (7)

main.cpp

```
  if( myPiggy.isFull() )
    cout << endl << "Das Sparschwein ist voll!" << endl;

  cout << "Das Sparschwein wird geschlachtet!" << endl;
  unsigned long cents = myPiggy.breakInto( c1, c10, c50, e1 );

  cout << "Im Sparschwein waren: "<< endl;

  showCoins( c1, c10, c50, e1 );

  cout << "Das sind " << cents / 100 << " Euros und " << cents % 100 << " cents."
       << endl << endl << "Gratuliere!" << endl;

  return 0;
}
```

Sparschwein ist voll?

Sparschwein wird aufgebrochen und gesparte Summe wird berechnet

```
C:\WINDOWS\system32\cmd.exe
Das Sparschwein ist leer!

Ersparnisse machen gluecklich!
Werfen Sie einige 1-Cent Stuecke ein: 123
Werfen Sie einige 10-Cent Stuecke ein: 220
Werfen Sie einige 50-Cent Stuecke ein: 77
Werfen Sie einige Euros ein: 90
Vielen Dank!

Das Sparschwein ist voll!
Das Sparschwein wird geschlachtet!
Im Sparschwein waren:
123 1-Cent Stuecke
220 10-Cent Stuecke
77 50-Cent Stuecke
80 1-Euro Stuecke
Das sind 141 Euros und 73 cents.

Gratuliere!
Drücken Sie eine beliebige Taste . . .
```

- Aufg. 5.02: OOP, Definition von Klassen

Aufg. 5.03: OOP, Klassendefinition Lsg. (1)

a)

```
class A {
  private:
    long secretKey;
  public:
    encode( const string& );
    decode( const string& );
  //...
};
```

Die Methoden **`encode( )`** und **`decode( )`** müssen einen Return-Typ aufweisen

b)

```
class B {
    long numerator, denominator;
  private:
    void convert( double );
    long gcd( void );
  //...
}
```

In der Klassendefinition fehlt die öffentliche Schnittstelle

Ein Semikolon hinter der schließenden Klammer **`}`** fehlt

Aufg. 5.03 : OOP, Klassendefinition Lsg. (2)

c)

```
class KFZ {
    string hers;
  public:
    ~KFZ();
    ~KFZ( string& );
  //...
};
```

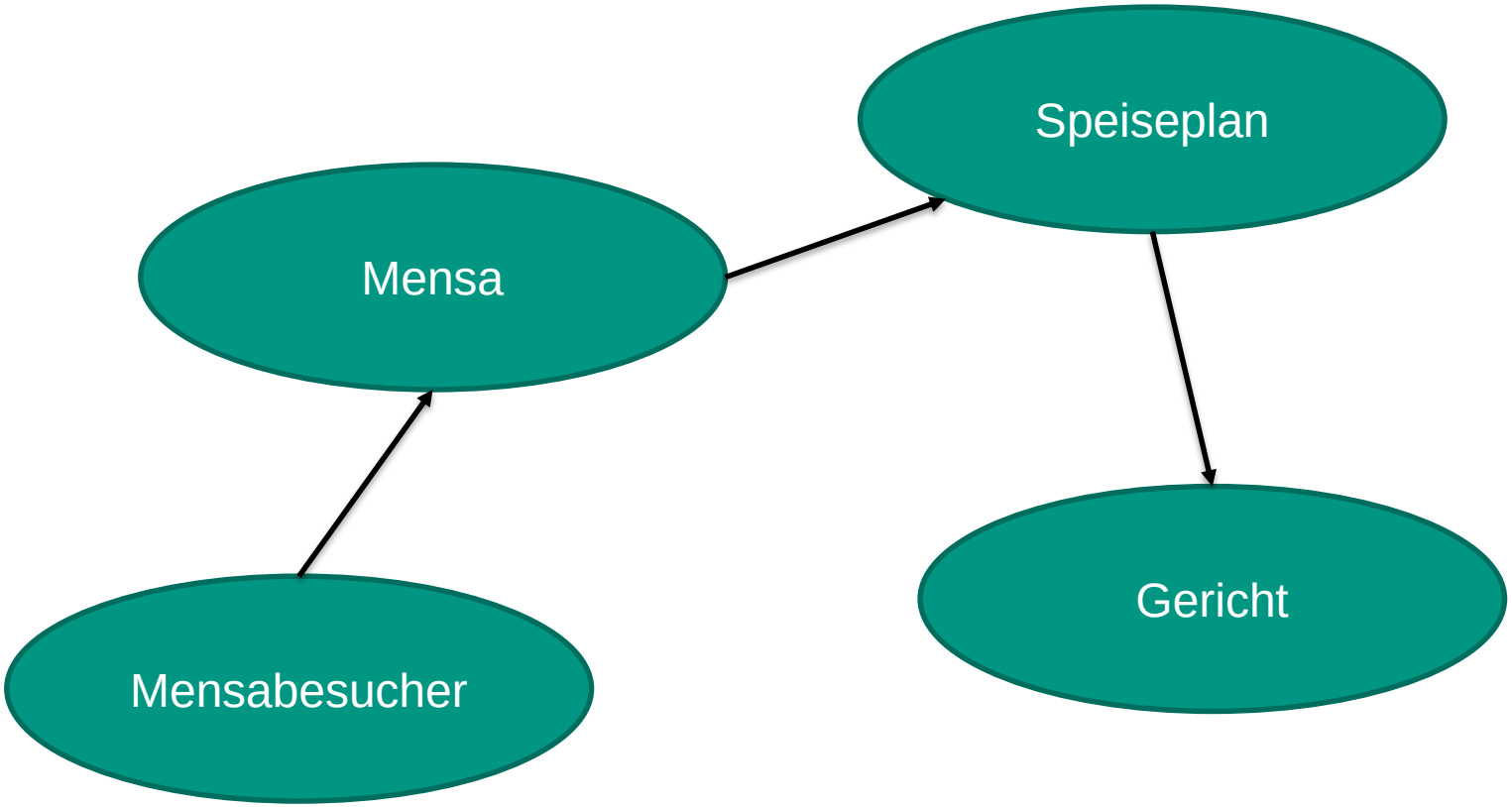
Der Destruktor einer Klasse hat keine Parameter und kann deshalb nicht überladen werden

- Aufg. 5.03: OOP, Klassendefinition

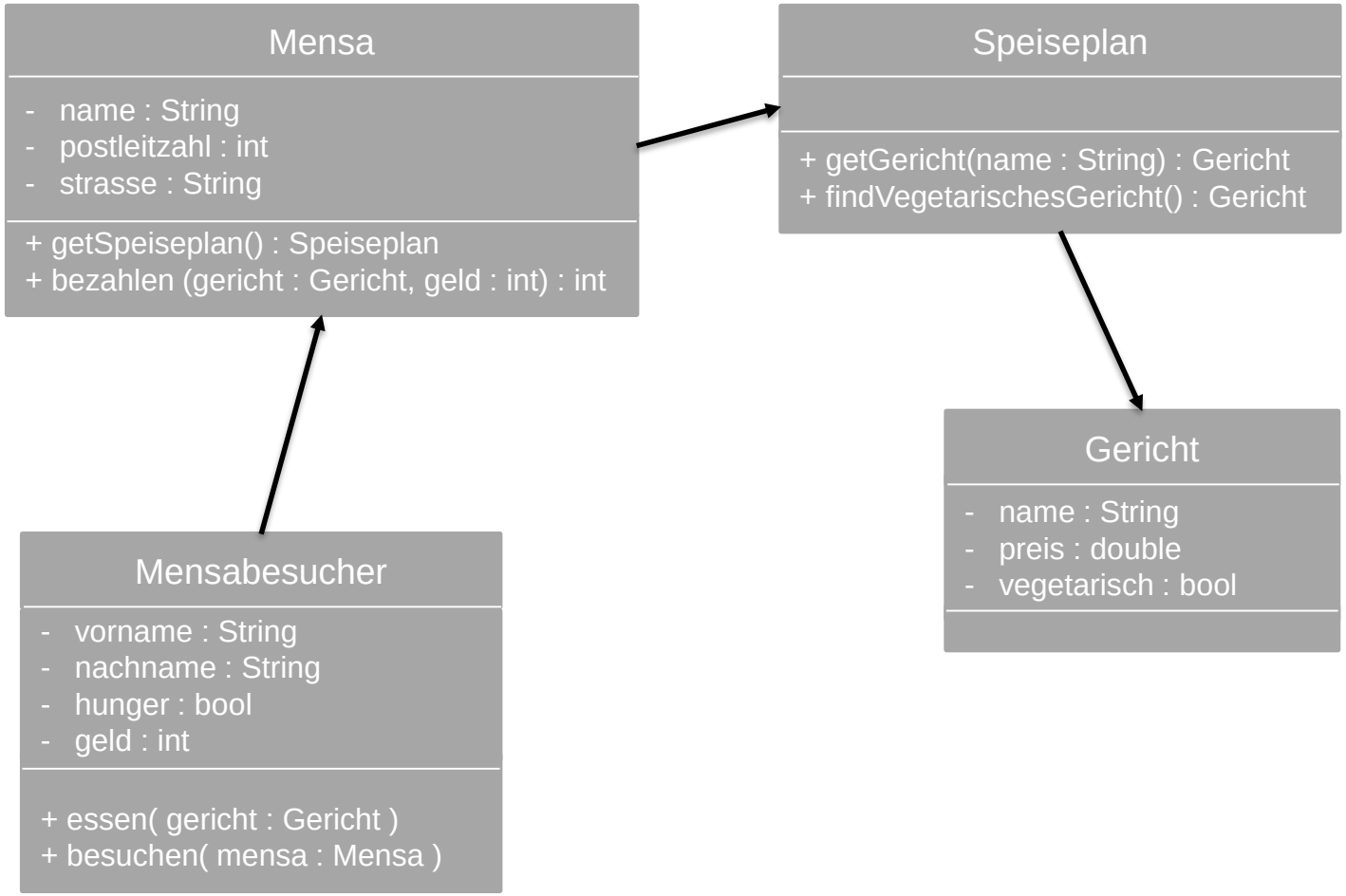
Aufg. 5.04: Modellierung mit Klassendiagrammen

- Stellen Sie sich vor, Sie möchten ein Startup gründen das mit einer neuen Software den Betrieb einer Mensa verbessern möchte. Dazu müssen Sie zuerst modellieren, wie die Mensa aktuell funktioniert.
- Überlegen Sie sich, welche Objekte in einer Mensa miteinander interagieren. Welche Klassen, Attribute und Methoden könnten geeignet sein, um diese Objekte zu beschreiben?
- Stellen Sie ein Klassendiagramm für die von Ihnen gewählten Klassen auf.
- Erstellen Sie den Programmcode zur Umsetzung des modellierten Klassendiagramms (Methoden nur deklarieren, nicht definieren).

Aufg. 5.04: Modellierung mit Klassendiagrammen Beispiel-Lsg.



Aufg. 5.04: Modellierung mit Klassendiagrammen Beispiel-Lsg.



Aufg. 5.04: Modellierung mit Klassendiagrammen Beispiel-Lsg.

```
class Mensa {
  private:
    String  name, strasse;
    int postleitzahl;
public:
    Speiseplan getSpeiseplan();
    int bezahlen (Gericht gericht , int geld );
};

class Mensabesucher {
 private:
    String vorname, nachname;
    bool hunger;
    int geld;
public:
    void essen (Gericht gericht);
    void besuchen (Mensa mensa );
};

class Speiseplan {
public:
    Gericht getGericht(String name);
    Gericht findVegetarischesGericht ();
};
```

```
class Gericht {
private:
    String name;
    double preis;
    bool vegetarisch;
};
```

Vielen Dank für Ihre Aufmerksamkeit

Daniel Grimm
Karlsruher Institut für Technology (KIT) – ITIV
daniel.grimm@kit.edu