**Dipl.Phys. Gerald Kempfer**

Berliner Hochschule für Technik
- University of Applied Sciences -
Fachbereich VI – Informatik und Medien
Studiengang Technische Informatik Bachelor

# Klausur zur Lehrveranstaltung "Objektorientierte Programmierung" 21. Juli 2023

**Hinweise:**

Die Teilnahme an der Klausur ist nur bei Bestehen der Übungsaufgaben zulässig!

Zum Bestehen der Klausur benötigen Sie 50 Punkte von 100 möglichen Punkten.

Die Bearbeitungszeit der Klausur beginnt pünktlich um 16.15 Uhr und endet – ebenfalls pünktlich – um 17.45 Uhr. Sie haben also 90 Minuten Zeit.

Folgende Hilfsmittel in Papierform sind zugelassen: Skripte, Mitschriften und Bücher. **Nicht** zugelassen sind elektronische Geräte (Handys, Notebooks, PDAs, usw.).

Schreiben Sie mit Kugelschreiber oder Füller, **nicht** mit Bleistift! Verwenden Sie **nicht** die Farbe Rot. Schreiben Sie leserlich – was ich nicht lesen kann, ist grundsätzlich falsch! Beschreiben Sie nur die Vorderseiten!

Jeder Austausch mit anderen Personen wird als Täuschungsversuch gewertet und führt dazu, dass die Klausuren aller Beteiligten als „nicht bestanden" gewertet werden.

Erläuterungen sollten kurz, aber dennoch präzise und vollständig sein. Wenn möglich ist eine stichpunktartige Beantwortung zu wählen, sofern die Verständlichkeit gegeben ist. Im Zweifelsfall können ganze Sätze Klarheit schaffen.

Kennzeichnen Sie jedes Blatt, das Sie abgeben, oben links mit Ihrer Matrikelnummer.

**Viel Erfolg!**

---

Name: _____

Matrikelnummer: _____ letzter Prüfungsversuch: ☐

Bewertung:

| Aufgabe | Mögliche Punktzahl | Erreichte Punktzahl |
|---|---|---|
| 1 | 15 | 14 |
| 2 | 30 | 30 |
| 3 | 15 | 15 |
| 4 | 16 | 16 |
| 5 | 24 | 24 |
| **Summe** | **100** | 99 |

Note Klausur 1,0

**Aufgabe 1:** Multiple-Choice-Fragen. (14 / 15)
Kreuzen Sie an, wie der Satz richtig heißen muss. Es gibt immer nur eine richtige Antwort.

`endl` ist
- ☐ eine Konstante.
- ☐ eine Systemvariable.
- ✓ ☒ ein Manipulator.

`this` ist
- ✓ ☒ ein Zeiger.
- ☐ ein Makro.
- ☐ eine Referenz.

Eine abstrakte Klasse kann
- ☐ nicht vererbt werden.
- ☐ zum Erzeugen von Objekten genutzt werden.
- ✓ ☒ eine oder mehrere rein virtuelle Methoden haben.

Eine Referenz ist
- ☐ eine Adresse.
- ☐ ein Zeiger.
- ✓ ☒ ein Aliasname.

Eine `friend`-Funktion darf
- ☐ auf private Elemente der befreundeten Klasse nur lesend zugreifen.
- ☐ nicht auf `protected`-Elemente der befreundeten Klasse zugreifen.
- ✓ ☒ auf alle Elemente der befreundeten Klasse zugreifen.

Eine Funktion ist überladen, wenn
- ☐ wenn sie im Programm mehrmals aufgerufen wird.
- ✓ ☒ wenn es weitere Funktionen mit dem gleichen Namen aber unterschiedlichen Parametern und/oder Rückgabedatentypen gibt.
- ☐ sie übergewichtig ist.

Ein Iterator ist
- ☐ ein Bruder vom Terminator.
- ☐ ein Zusatzprogramm zum Compiler.
- ✓ ☒ ein Konzept.

Der Datentyp für Wahrheitswerte (benannt nach George Boole) heißt
- ✓ ☒ `bool`.
- ☐ `boole`.
- ☐ `boolean`.

`: :` ist ein
- ☐ Beendigungsoperator.
- ☐ Begleichungsoperator.
- ✓ ☒ Bereichsoperator.

Templates werden auch
- ☐ temporäre Typen genannt.
- ☐ genetische Typen genannt.
- ☒ generische Typen genannt.

Der Kopier-Konstruktor hat
- ☐ einen Parameter weniger als der Standard-Konstruktor.
- ☐ genauso viele Parameter wie der Standard-Konstruktor.
- ☒ einen Parameter mehr als der Standard-Konstruktor.

Der Destruktor einer Klasse
- ☐ kann einen beliebigen Namen haben.
- ☒ beginnt mit einer Tilde (~).
- ☐ kann überladen werden.

Wird bei der Vererbung einer Klasse (`class`) kein Zugriffsattribut angegeben, wird automatisch das Zugriffsattribut
- ☒ `private` verwendet.
- ☐ `protected` verwendet.
- ☐ `public` verwendet.

Eine Referenzvariable
- ☒ muss bei der Definition initialisiert werden.
- ☐ darf bei der Definition initialisiert werden, aber muss nicht.
- ☐ darf bei der Definition auf keinen Fall initialisiert werden.

`this`
- ☒ wird automatisch definiert.
- ☒ kann in allen Methoden verwendet werden. nicht in static-Methoden!
- ☐ kann nur in `public`-Methoden verwendet werden.

**Aufgabe 2:** **a)** Was gibt das folgende Programm aus? Schreiben Sie die Ausgaben auf das nächste Blatt! ( 23 / 23 )

```
#include <iostream>

using namespace std;

class Transportmittel
{  protected:
      double Geschwindigkeit;
   public:
      Transportmittel()          { cout << "+T" << endl; }
      virtual ~Transportmittel() { cout << "-T" << endl; }
};

class Landtransportmittel: virtual public Transportmittel
{  public:
      int AnzahlRaeder;
   public:
      Landtransportmittel()          { cout << "+L" << endl; }
      virtual ~Landtransportmittel() { cout << "-L" << endl; }
};

class Wassertransportmittel: virtual public Transportmittel
{  public:
      int AnzahlRuder;
      Wassertransportmittel()  { cout << "+W" << endl; }
      ~Wassertransportmittel() { cout << "-W" << endl; }
};

class Amphibienfahrzeug: public Landtransportmittel,
                         public Wassertransportmittel
{     int AnzahlTueren;
   public:
      Amphibienfahrzeug()  { cout << "+A" << endl; }
      ~Amphibienfahrzeug() { cout << "-A" << endl; }
};

int main()
{
   Transportmittel *Porsche = NULL;
   Wassertransportmittel Motorboot;
   Landtransportmittel *Dreirad;
   Amphibienfahrzeug Watercar;

   cout << "Textausgabe" << endl;
   Dreirad = new Amphibienfahrzeug;
   Porsche = new Landtransportmittel;

   // Zeilen für den zweiten Teil der Aufgabe:
   // Porsche->Geschwindigkeit = 329.0;
   // Motorboot.AnzahlRuder = 0;
   // Watercar.Untergegangen = true;
   // Watercar.AnzahlRuder = 2;
   // Porsche->AnzahlRaeder = 4;
   // Dreirad->AnzahlRaeder = 3;
   // Watercar->AnzahlRaeder = 6;
   // Ende Zeilen für zweiten Teil der Aufgabe

   delete Dreirad;

   return 0;
}
```

Ausgabe des oben stehenden Programms:

+T
+W
+T
+L
+W
+A
Textausgabe
+T
+L
+W
+A
+T
+L
-A
-W
-L
-T
-A
-W
-L
-T
-W
-T

**b)** Wenn die folgenden Zeilen, die in dem oben stehenden Hauptprogramm auskommentiert sind, eingefügt werden würden, welche Zeilen sind korrekt und welche sind nicht korrekt? ( 7 / 7)

| | | |
|---|---|---|
| `Porsche->Geschwindigkeit = 329.0;` | // ☐ korrekt | ☒ nicht korrekt ✓ |
| `Motorboot.AnzahlRuder = 0;` | // ☒ korrekt | ☐ nicht korrekt ✓ |
| `Watercar.Untergegangen = true;` | // ☐ korrekt | ☒ nicht korrekt ✓ |
| `Watercar.AnzahlRuder = 2;` | // ☒ korrekt | ☐ nicht korrekt ✓ |
| `Porsche->AnzahlRaeder = 4;` | // ☐ korrekt | ☒ nicht korrekt ✓ |
| `Dreirad->AnzahlRaeder = 3;` | // ☒ korrekt | ☐ nicht korrekt ✓ |
| `Watercar->AnzahlRaeder = 6;` | // ☐ korrekt | ☒ nicht korrekt ✓ |

**Aufgabe 3:** Gegeben sind vier Klassen sowie vier Eigenschaften:

| **Klasse** | **Eigenschaft** |
|---|---|
| Vierbeiner (VB) | Name (na, `string`) |
| Lebewesen (LE) | Anzahl Flügel (af, `int`) |
| Hund (HU) | Anzahl Beine (ab, `int`) |
| Tier (TI) | Gewicht (ge, `double`) |

Erstellen Sie aus diesen Klassen eine Klassenhierarchie: Welche Klasse ist die Basisklasse und welche werden in welcher Reihenfolge abgeleitet? Und welche Eigenschaft gehört zu welcher Klasse?
Definieren Sie dazu im unten stehenden Quelltext **passend zum Hauptprogramm** (siehe nächste Seite) die Klassen mit den **privaten** Eigenschaften und mit jeweils einem **Konstruktor**, der per **Initialisierungsliste** den Konstruktor der Oberklasse aufruft und die Eigenschaft(en) setzt. Dabei muss nicht jede Klasse zwingend eine Eigenschaft besitzen.
Durch Verwendung der in Klammern gesetzten Abkürzungen können Sie sich Schreibarbeit ersparen. ( 15 / 15 )

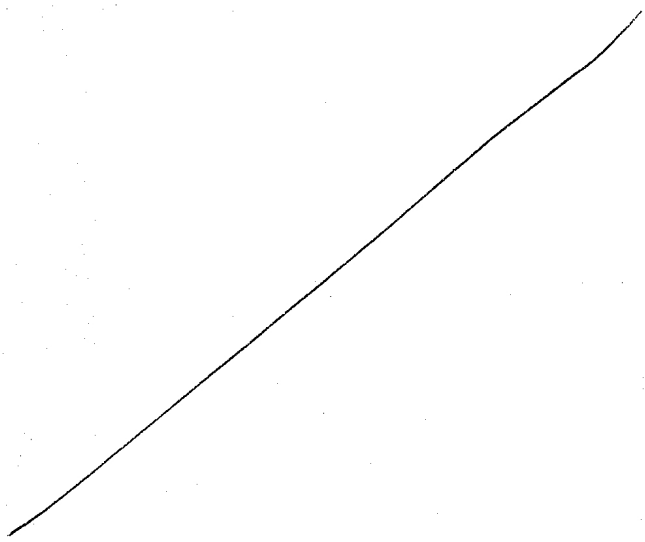
```
#include <iostream>

using namespace std;

// Platz für die vier Klassen:
```

// Fortsetzung auf der nächsten Seite

```
// Fortsetzung der vier Klassen für Aufgabe 3:
class LE {
  double ge;
  public: LE(double ge): ge(ge) {};
};
class TI: public LE {
  int ab, af;
  public: TI(int ab, int af, double ge): LE(ge), ab(ab), af(af) {};
};
class VB: public TI {
  public: VB(int af, double ge): TI(4, af, ge) {};
};
class HU: public VB {
  string Na;
  public: HU(double ge, string Na): VB(0, ge), Na(Na) {};
};
```



```
int main()
{
   LE aMouse(              // LE steht für Lebewesen
             0.204);       // Gewicht (ge)
   TI aBird(               // TI steht für Tier
             2,            // Anzahl Beine (ab)
             2,            // Anzahl Flügel (af)
             0.143);       // Gewicht (ge)
   VB Pegasus(             // VB steht für Vierbeiner
             2,            // Anzahl Flügel (af)
             137.5);       // Gewicht (ge)
   // Anmerkung: Pegasus ist in der griechischen
   //            Mythologie ein geflügeltes Pferd
   HU aDog(                // HU steht für Hund
             23.8,         // Gewicht (ge)
             "Rex");       // Name (na)
}
```

**Aufgabe 4:** Definieren Sie zwei Funktions-Templates passend zu Hauptprogramm:
Das erste Funktions-Template mit dem Namen `copyArrayToVector` soll die Elemente eines Arrays in ein Vektor kopieren. Als Parameter werden das Array, die Anzahl der Arrayelemente sowie eine Referenz auf den Vektor übergeben. Zurückgegeben wird nichts. (7 Punkte)
Das zweite Funktions-Template mit dem Namen `printVector` soll alle Elemente des Vektors mit Hilfe einer `for`-Schleife und eines **Iterators** hintereinander ausgeben (es soll **keine for-Range-Schleife** verwendet werden). Als Parameter wird eine Referenz auf den Vektor übergeben; es gibt keinen Rückgabewert. (9 Punkte) ( 16 / 16 )

**Ausgabe des Programmes:**

```
12345678910
Hallo Welt
```

**Programm:**

```
#include <iostream>
#include <vector>

using namespace std;

// Platz für Ihre Definitionen der Funktions-Templates:
template<class T> void copyArrayToVector
(T *orig, int size, vector<T> &dest){
   for(int i=0; i<size; i++){
      dest.push_back(*(orig+i));
   }
}
template<class T> void printVector( vector<T> &vec){
   for(vector<T>::iterator it = vec.begin();
       it != vec.end(); it++){
      cout << *it << endl;
   }
}

int main()
{
   int Zahlen[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
   string Text("Hallo Welt");
   vector<char> Zeichenvector;
   vector<int> Zahlenvector;

   copyArrayToVector(Zahlen, 10, Zahlenvector);
   copyArrayToVector(Text.c_str(), Text.size(), Zeichenvector);

   printVector(Zahlenvector);   cout << endl;
   printVector(Zeichenvector);  cout << endl;

   return 0;
}
```

**Aufgabe 5:** In der Schule haben Sie bestimmt gelernt, dass Äpfel und Birnen nicht addiert werden können. Hier können Sie das Gegenteil beweisen:
In den folgenden Klassen fehlen noch die Methoden bzw. die Funktionen, damit das Hauptprogramm korrekt laufen kann. Ergänzen Sie die Klasse `cObst` im Namensraum `nObst` um die nötigen Operatoren, um
- Obst zu addieren sowie
- Obst und Gemüse zu addieren.

Es wird jeweils das Gewicht addiert. Dabei ergibt die Addition von Obst und Gemüse wieder ein Objekt der Klasse `cObst`.
Ferner muss noch der Ausgabeoperator in der Klasse `cObst` überladen werden. Dabei soll das Gewicht vom Obst bzw. vom Obst-Gemüse-Gemisch ausgegeben werden (wie bei der Ausgabe des Programms vorgegeben!).
Beachten Sie dabei die Namensräume! Die Namensräume sollen nur erweitert werden (d.h. **es darf kein `using` verwendet werden**)! ( 24 / 24 )

**Ausgabe des Programmes:**

```
Obst: 6 kg
Obst und Gemuese: 8.3 kg
Obst und Gemuese: 10.1 kg
```

**Vorgegebenes Programm:**

```
#include <iostream>

using namespace std;

namespace nGemuese
{
   class cGemuese; // Vorwaertsdeklaration
}

namespace nObst
{
   class cObst
   {
         double Gewicht;
         bool Gemischt; // Obst und Gemuese gemischt
      public:
         cObst(double Gew = 0.0, bool Gem = false)
         : Gewicht(Gew), Gemischt(Gem) { }
         double getGewicht() { return Gewicht; }
         bool getGemischt()  { return Gemischt; }
         // Platz für Ihre Deklarationen (ohne Definitionen; 6 Punkte):
         cObst operator+( cObst &summand);
         cObst operator+(nGemuese::cGemuese &summand);
         friend ostream &operator<<(ostream &ostr, cObst &obj);

   }; // class cObst
} // namespace nObst
```

```
namespace nGemuese
{
   class cGemuese
   {
         double Gewicht;
      public:
         cGemuese(double Gew = 0.0)
         : Gewicht(Gew) { }
         double getGewicht() { return Gewicht; }
   }; // class cGemuese
} // namespace nGemuese

int main()
{
   nObst::cObst Aepfel(3.5), Birnen(2.5), Bananen(1.8);
   nObst::cObst Obstsalat, Salat;
   nGemuese::cGemuese Bohnen(2.3);

   Obstsalat = Aepfel + Birnen;   cout << Obstsalat << endl;
   Salat = Obstsalat + Bohnen;    cout << Salat     << endl;
   Obstsalat = Salat + Bananen;   cout << Obstsalat << endl;

   return 0;
}

// Platz für Ihre Definitionen (18 Punkte):
```

namespace nObst {

ostream &operator<<
(ostream &ostr, cObst &obj) {
cout << "Obst";
if(obj.getGemischt()) {
cout << " und Gemuese"; }
ostr << ": " << obj.getGewicht() << "kg" << endl;
return ostr;
}

✓

```
// Fortsetzung der Definitionen für Aufgabe 5:
cObst cObst::operator+(cObst &Summand) {
    cObst c(this->getGewicht() + Summand.getGewicht(),
            this->getGewicht() || Summand.getGewicht);
    return c;
}
cObst cObst::operator+
    (nGemuese::cGemuese &Summand) {
    cObst c (this->getGewicht() + Summand.getGewicht(),
             true);
    return c;
}
//
} // Ende nObst
```