Objekte / Objektorientierte Programmierung Klausur WS 2018 / 2019 (mit Lösungen und Bewertungsschema)

Aufg. Nr.	Text	Antwort/Lösung	Pkt
1.	Was unterscheidet Strukturen von Klassen? Was charakterisiert die 3 Stufen des objektorientierten Vorgehens nach Wegner? Nennen Sie zu jeder Stufe zwei Schlüsselbegriffe: 1. Objekt-basiertes Vorgehen 2. Klassen-basiertes Vorgehen 3. Objekt-orientiertes Vorgehen	2P: - Bei Structs sind die Datenmitglieder standardmäßig „public“, bei Klassen „private“ - Bei Klassen gibt es Funktionsmitglieder (Methoden) - Bei Klassen gibt es die Vererbung je 2P: 1. Daten- und Funktionskapselung 2. Klassenbildung (Abstraktion / Instanziierung) 3. Vererbung und Polymorphismus	8

2.	Folgende Klasse ist vorgegeben:
	
	`class cDemo1 {`
	`  int cnt, max;`
	`  double amount [12];`
	`};`
	
	Was wird in jeder der folgenden Definitionen definiert?
	
	`cDemo1 * ptr = new cDemo1 [64];`
	
	`cDemo1 * ptr = new cDemo1 (64);`
	
	`cDemo1 * ptr [64] = { new cDemo1 };`
		je 2P:
		
		`K6 * ap = new cDemo1 [64];`
		
		Erzeugt einen Vektor mit 7 Objekten, es wird der Standard-Konstruktor verwendet. ap zeigt danach auf ein Feld aus 7 Instanzen der Klasse K6, es wird der Standardkonstruktor (ohne Parameter) aufgerufen. Die Klammern sind der Subskript-Operator.
		
		`K6 * ap = new cDemo1 (64);`
		
		Erzeugt 1 Objekt und startet den Konstruktor mit dem Parameterwert 7. Das Objekt auf das ap zeigt wird eine Instanz der Klasse K6 und der Konstruktor wird mit dem Parameterwert 7 aufgerufen Die Klammern sind der Aufrufoperator.
		
		`K6 * ap [64] = { new cDemo1 };`
		
		Erzeugt einen Vektor aus 7 Zeigern auf Objekte der Klasse K6, wobei der erste durch die Initialisierungsliste mit dem Zeiger auf ein neu erzeugtes Objekt belegt wird, die anderen Zeiger bleiben ungesetzt.
			6

3.	Multiple Choice Aufgaben, es können keine, eine oder mehrere Aussagen der Auswahlen richtig sein. Kreuzen Sie nur die richtigen Aussagen an. a) abstrakte Klassen: • Eine abstrakte Klasse erzeugt von jeder Instanz nur die indirekte Referenz. • Konkrete Klassen haben Instanzen, abstrakte Klassen nicht. • Von einer abstrakten Klasse kann genau 1 Instanz erzeugt werden. • Eine konkrete Klasse wird in C++ mittels abstrakter, rein virtueller Funktionen implementiert. • Eine vollständig überdeckte Klasse ist eine abstrakte Klasse. • Keine der Aussagen trifft zu. b) Beziehungen von Objekten: • Durch eine Reproduktion der Kardinalitäten können wir 1 * M Beziehungen in konditional komplexe Beziehungen umwandeln. • Unter der Kardinalität versteht man die Position der Klasse in der Klassenhierarchie. • Die Kardinalität gibt an, mit wie vielen Objekten anderer Klassen Beziehungen existieren. • Bei einer konditional einfachen Beziehung hat ein Objekt keine Beziehung zu einem anderen Objekt oder eine Beziehung zu genau einem Objekt. • N * M Beziehungen werden als komplexe Beziehungen bezeichnet. • Keine der Aussagen trifft zu. c) Referenzen: • Eine Referenz ist ein Alias-Name auf ein Objekt und bleibt ständig mit ihm verbunden. • Die Einführung von Referenzen erhöht den Laufzeitaufwand für Methodenaufrufe. • Die Einführung von Referenzen reduziert den Laufzeitaufwand für Methodenaufrufe.	Bewertung: jeweils 3 Punkte für einen korrekten MC-Sechserblock; bei 1 Fehler 2 P, bei 2 Fehler 1 P, bei >= 3 Fehler 0 P (Alternativ: Für jede richtige Teilfrage 0,5 Punkte) *a) F – R – F – F – R – F* *b) F -- F -- R -- R -- R -- F* *c) R -- F -- R -- R -- R – F*	12

	• Werden Attributswerte in einer Referenz auf ein Objekt verändert, so wird damit auch das Objekt verändert. • Gibt eine Funktion eine Referenz auf ein Objekt zurück, dann muss dieses Objekt über das Verlassen der Funktion hinaus existieren. • Keine der Aussagen trifft zu. d) Polymorphismus / „late binding“: • Die zuletzt fehlerfrei übersetzte Version einer Funktion wird an eine abstrakte Instanz gebunden. • Zur Laufzeit des Programms wird entschieden, welche Überladung einer virtuellen Funktion angesprungen wird. • Zur Laufzeit des Programms wird entschieden, welche Überschreibung einer virtuellen Funktion angesprungen wird. • „late binding“ ist die Voraussetzung für den Polymorphismus. • „late binding“ bedeutet, dass die überschriebene Version einer virtuellen Funktion erst zur Laufzeit ausgesucht wird, passend zum aktuellen Objekttyp. • Keine der Aussagen trifft zu.	*d) F -- F -- R -- R -- R -- F*	

4.	a) Finden Sie mindestens 6 Fehler im folgenden Programm und Beschreiben Sie jeden mit wenigen Worten.
	b) Beschreiben Sie die Funktion des Programms (ignorieren Sie dabei die Fehler).
	
	`#include <iosteam>`
	`using homespace std;`
	
	`class cSchiff {`
	`public:`
	`  virtual void anlegen () == 0;`
	`};`
	
	`class cTanker : private cSchiff {`
	`fabric:`
	`  virtual void anlegen () {`
	`  cout >> "Tanker legt an mit Schleppern" <<`
	`endl;`
	`  }`
	`};`
	
	`class cUboot : public cBoot {`
	`public:`
	`  virtual void anlegen (cSchiff& s) {`
	`  cout << "Uboot taucht auf und legt an" <<`
	`endl;`
	`  }`
	`}`
	
	`int main () {`
	`  cSchiff * sp;`
	`  cTanker t1;`
	`  cUboot u1;`
	
	`  sp = &t1;`
	`  sp->anlegen();`
	
	`  sp = *u1;`
	`  sp.anlegen();`
	
	`  return 0;`
	`}`
		a) max 6P (0,5P je Fehler wenn keine Erklärung)
		b) Erkennen Polymorphismus: 4P
		
		`// Kleines Beispiel Polymorphismus`
		
		`#include <iostream>`
		`using homespace std;`
		
		`class cSchiff {`
		`public:`
		`  virtual void anlegen () == 0;`
		`};`
		
		`class Tanker : private cSchiff {`
		`fabric:`
		`  virtual void anlegen () {`
		`  cout >> "Tanker legt an mit Schleppern" << endl;`
		`  }`
		`};`
		
		`class cUboot : public cBoot {`
		`public:`
		`  virtual void anlegen (cSchiff& s) {`
		`  cout << "Uboot taucht auf und legt an" << endl;`
		`  }`
		`};`
		
		`int main () {`
		`  cSchiff * sp;`
		`  cTanker t1;`
		`  cUboot u1;`
		
		`  sp = &t1;`
		`  sp->anlegen();`
		
		`  sp = *u1;`
		`  sp.anlegen();`
		
		`  return 0;`
		`}`
			10

5.	C++ Programmierung
	
	Schreiben Sie folgendes Programmstück so um, dass beim Funktionsaufruf sowie beim Return **kein** Kopierkonstruktor aufgerufen wird:
	
	`class AT {`
	`  int i;`
	
	`public:`
	`  AT  Doit (AT  in1) {`
	`  AT out1;`
	
	`  out1.i = in1.i % 2;`
	`  return out1;`
	`  }`
	`};`
	
	`int main () {`
	`  AT a, b, c;`
	`  c = a.Doit(b);`
	
	`  return 0;`
	`}`
		1 P je Referenz ***&***
		2 P für das ***static***
		
		`class AT {`
		`  int i;`
		
		`public:`
		`  AT & Doit (AT & in1) {`
		`  static  AT out1;`
		
		`  out1.i = in1.i % 2;`
		`  return out1;`
		`  }`
		`};`
		
		`int main () {`
		`  AT a, b, c;`
		`  c = a.Doit(b);`
		
		`  return 0;`
		`}`
			4

6.

Programmieraufgabe am Rechner. Verwenden Sie MS Windows und das MS Visual Studio. Beachten Sie die **Anleitung für die Programmieraufgabe** im Anhang der Klausur.

Aufgabe: Programm zur Kreisaddition.

- Vorgegeben ist folgende Klasse cPunkt mit Attributen und Methoden:

```
class cPunkt {
        double x;       // X-Koordinate
        double y;       // Y-Koordinate

public:
        cPunkt (double x_in = 0.0, double y_in = 0.0) {
                x = x_in; y = y_in;
        }

        void ausgabe() {
                cout << "Punkt: " << x << " / " << y << endl;
        }

        friend ostream& operator << (ostream&, const
cPunkt&);
};
```

- Ergänzen Sie die Implementierung des Ausgabeoperators der Klasse cPunkt.

- Erstellen Sie eine Klasse cKreis mit den Attributen Mittelpunkt und Durchmesser (Diameter). Der Mittelpunkt soll als Variable der Klasse cPunkt realisiert werden (Aggregation).

- Programmieren Sie einen universellen Konstruktor für die Klasse cKreis und ergänzen Sie die Konstruktorenkaskade für

```
// Programmieraufgabe Kreisaddition

#include <iostream>
#include <math.h>   // wg. sqrt()
using namespace std;

#define PI 3.14195    // Hilfsdefinition wenn nicht math.h

// Hilfsrechnung:
// F = PI * r * r
// r = sqrt(F/PI)

class cPunkt {
        double x;       // X-Koordinate
        double y;       // Y-Koordinate

public:
        cPunkt (double x_in = 0.0, double y_in = 0.0) {
                x = x_in; y = y_in;
        }

        void ausgabe() {
                cout << "Punkt: " << x << " / " << y << endl;
        }

        friend ostream& operator << (ostream&, const cPunkt&);
};

ostream& operator << (ostream& o, const cPunkt& p) { //
Ausgabeoperator cPunkt
        o << "Punkt: " << p.x << " / " << p.y;
        return o;
}


class cKreis {
        cPunkt mitte;
        double durchmesser;
        double flaeche() const {
```

60

die Variable der Klasse cPunkt.

- Programmieren Sie eine private Methode ***flaeche()*** zur Flächenberechnung. Die Formel zur Flächenberechnung ist vorgegeben, in folgender Notation:

  Fläche = PI * ((1/2 Durchmesser) ^{2})

  Diese Notation müssen Sie in C++-Syntax umsetzen um die Flächenberechnung korrekt programmieren zu können.

- Erstellen Sie eine Ausgabemethode ***ausgabe()*** für die Klasse cKreis. Dabei sollen der Mittelpunkt, der Durchmesser und die Fläche ausgegeben werden.

- Ergänzen Sie einen Operator **+** für die Addition von 2 Kreisen: Dabei wird die Summe der Flächen der beiden Kreise gebildet. Neuer Mittelpunkt wird der Mittelpunkt des größeren der beiden Kreise. Der größere Kreis „schluckt“ also den kleineren Kreis und wird dadurch noch größer. Aus der neuen Fläche des Ergebniskreises müssen Sie den Durchmesser des Ergebniskreises berechnen.
  Tipp: Hilfszeichnung und Hilfsrechnung machen.

- Programmieren Sie zusätzlich einen Ausgabeoperator **<<** für cKreis.
  (Tipp: Ausgabeoperator der Klasse cPunkt mitverwenden)

- Vorgegeben ist folgendes Hauptprogramm, um Ihre Programmierung zu überprüfen:

```
int main() {
        cKreis kreis1 (1.3, 2.6, 7.1); // x, y des Mittelpunktes und Durchmesser
        cKreis kreis2 (3.4, 5.6, 17.7);
        cKreis kreis3;
```

```
                return (PI*(durchmesser/2.0)*(durchmesser/2.0));
        }

public:
        // Konstruktor braucht 3 Werte, gibt 2 davon für die Konstruktion des Mittelpunkts weiter

        cKreis (double x_in = 0.0, double y_in = 0.0, double d_in = 0.0) : mitte (x_in, y_in) {
                durchmesser = d_in;
        }

        void ausgabe() {
                cout << "Kreis: Mittelpunkt ist "; mitte.ausgabe();
                cout << " Durchmesser: " << durchmesser << " Flaeche: " << flaeche() << endl;
        }

        cKreis & operator + (const cKreis & k2) {

                static cKreis help;

                // z.B.: Flächen addieren, Durchmesser neu berechnen,
                // Mittelpunkt wird der des größeren Operanden

                if (k2.durchmesser > durchmesser) {
                        help = k2;
                }
                else {
                        help = *this;
                }

                help.durchmesser = sqrt((flaeche()+k2.flaeche())/PI)*2.0;
                                // radius verdoppeln zum Durchmesser

                return help;
        }

        friend ostream& operator << (ostream&, const cKreis&);
```

```
        kreis1.ausgabe();
        kreis2.ausgabe();

        kreis3 = kreis1 + kreis2;
        kreis3.ausgabe();

        cout << "Ausgabeoperator kreis3: " << kreis3 <<
    endl;

        return 0;
    }
```

- Setzen Sie die Lösung mittels Artefaktetrennung um; für jede Klasse soll es eine Header-Datei mit der Klassendefinition sowie eine cpp-Datei mit der Implementierung der Methoden geben; zusätzlich eine Quelldatei **haustier.cpp** für das Hauptprogramm.

```
Bewertungsschema:                                    Punktzahl

- Programm laesst sich ohne Fehler/Warnungen uebersetzen 8P
- Codeformatierung und sinnvolle Kommentare               4P
- Klassendefinitionen und Konstruktoren                   6P
- Ueberladung der Ausgabeoperatoren <<                    6P
- Ueberladung Additions-Operator cKreis                  16P
- korrekte Berechnung Flaeche                             4P
- Hauptprogramm                                           6P
- Artefaktetrennung                                      10P

Summe:                                                   60P

Handskizzen können bei der Punktevergabe berücksichtigt werden
```

Platz für Ihre Handskizzen und Notizen (Sie können auch die Blattrückseiten verwenden):

```
};

ostream& operator << (ostream& o, const cKreis& k) { //
Ausgabeoperator verwendet Ausgabeoperator cPunkt
        o << "Kreis: Mittelpunkt ist " << k.mitte << " Durchmesser: "
                        << k.durchmesser << " Flaeche: "  << k.flaeche();
        return o;
}

int main() {
        cKreis kreis1 (1.3, 2.6, 7.1); // x, y des Mittelpunktes und
Durchmesser
        cKreis kreis2 (3.4, 5.6, 17.7);
        cKreis kreis3;

        kreis1.ausgabe();
        kreis2.ausgabe();

        kreis3 = kreis1 + kreis2;
        kreis3.ausgabe();

        cout << "Ausgabeoperator kreis3: " << kreis3 << endl;

        return 0;
}

/* Ausgabe:
Kreis: Mittelpunkt ist Punkt: 1.3 / 2.6
 Durchmesser: 7.1 Flaeche: 39.5964
Kreis: Mittelpunkt ist Punkt: 3.4 / 5.6
 Durchmesser: 17.7 Flaeche: 246.085
Kreis: Mittelpunkt ist Punkt: 3.4 / 5.6
 Durchmesser: 19.0709 Flaeche: 285.682
Ausgabeoperator kreis3: Kreis: Mittelpunkt ist Punkt: 3.4 / 5.6
Durchmesser: 19.0709 Flaeche: 285.682
*/
```