Klausur zu Objektorientierter Softwareentwicklung in C++

4. Februar 2003 (WS 2002/2003)

Beispiellösung

Bearbeitungszeit: 120 Minuten

NICHT MIT BLEISTIFT SCHREIBEN!!!

Name:	
Vorname:	
Matrikelnummer:	

Aufgabe	Punkte	Bewertung
1	**20**	
2	**10**	
3	**10**	
4	**10**	
5	**10**	
6	**10**	
(a)	1	
(b)	3	
(c)	6	
7	**10**	
(a)	5	
(b)	5	
Summe:	**80**	

Bitte benutzen Sie für die Lösungen den freigelassenen Platz nach der jeweiligen Angabe oder die Rückseite unter Angabe der Aufgabe. Nennen Sie möglichst alle impliziten Annahmen, die Sie treffen. `#include`-Anweisungen können aber der Einfachheit halber weggelassen werden!

Vollen Erfolg!!!

Aufgabe 1 **(je 2 Punkte pro Frage = 20 Punkte)**

Bitte jeweils **eine** Antwort ankreuzen (je nach Fragestellung die **richtige** oder **falsche**):

(1) Für die Implementierung eines TCP-Servers mit Qt benötigen Sie u.a. die Klasse:
A) `QSocketServer`
B) `QServerSocken`
$\boxed{\text{C)}}$ `QSocket`
D) `QPaintBrush`

(2) Eine Klasse heißt abstrakt, wenn
A) kein reelles Gegenstück denkbar ist
$\boxed{\text{B)}}$ sie mindestens eine rein virtuelle Methode besitzt
C) sie keinen konkreten Algorithmus enthält
D) niemand weiss, wozu man sie gebrauchen kann.

(3) Polymorphie tritt **nicht** auf bei
$\boxed{\text{A)}}$ Objektvariablen,
B) Objektreferenzen,
C) Objektzeigern,
D) Zeigern auf Objekte von abgeleiteten Klassen.

(4) Die Schnittstelle einer Klasse besteht aus
A) `polymorphen` Ringen,
B) den `protected`-Elementen einer Klasse oder
$\boxed{\text{C)}}$ den `public`-Elementen einer Klasse oder
D) konzentrischen Kreisen.

(5) Nach der Deklaration von `const int *p;` lässt der Compiler das folgende Statement noch zu:
A) `*p = 3`,
$\boxed{\text{B)}}$ `p++`,
C) `(*p)++`,
D) `int *q = p`.

(6) Damit Objekte einer Klasse in einer `map` verwaltet werden können, muss für Objekte dieser Klasse der
$\boxed{\text{A)}}$ <-Operator,
B) ==-Operator
C) `[]`-Operator
D) OP-Pfleger
überladen werden?

(7) Welche Aussage ist **falsch**: Wenn man bei einer eigenen von `QWidget` abgeleiteten Klasse auf das Drücken der linken Maustaste reagieren will, so muss man
A) die Methode `mousePressEvent(...)` redefinieren,
$\boxed{\text{B)}}$ das Signal `mousePressEvent(...)` mit einem Slot verbinden,
C) in Erfahrung bringen, welche Parameter `mousePressEvent()` hat,
D) Qt auf dem Rechner richtig installiert haben.

(8) Alle darstellbaren Fenster-Elemente von Qt sind Unterklassen von
A) `QGadget`,
B) `QWindowElement`
$\boxed{\text{C)}}$ `QWidget`
D) `QWindow`

(9) Wodurch können Qt-Signale gesendet werden:
A) mit `QObject::connect(...)`,
B) mit `QTrumpet::tuutTuut(...)`
$\boxed{\text{C)}}$ mit dem Qt-Makro `emit`
D) mit `QSocket::sendQtSignal(...)`

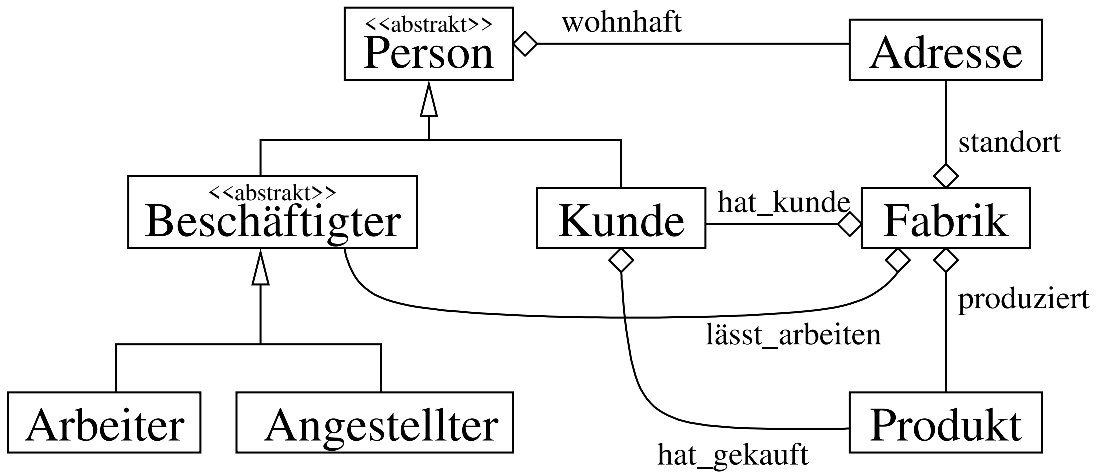
(10) Was bewirkt die Methode `QApplication::setMainWidget(...)`?
$\boxed{\text{A)}}$ Definiert ein übergebenes Widget als (Haupt-)Fenster,
$\boxed{\text{B)}}$ beendet die Anwendung beim Schließen dieses Widgets,
C) lässt den Main im Widget aus QApplication fliessen,
D) erreicht eine irreversible polymorphe Abstraktion von `QApplication`.

Aufgabe 2 **(10 Punkte)**

Zeichnen Sie ein UML-Klassendiagramm, das die nachfolgenden Begriffe sinnvoll, und analog zur realen Welt, in Beziehung setzt. Auf die Angabe von Methoden und Variablen kann verzichtet werden. Kandidaten für abstrakte Klassen sollten erkannt und gekennzeichnet werden, Assoziationen zwischen den Klassen auch:

Fabrik, Arbeiter, Angestellter, Beschäftigter, Kunde, Produkt, Adresse

Lösung:



Aufgabe 3 **(10 Punkte)**

Gegeben ist folgendes Programm:

```
class Life {
    int val;
public:
    Life(int v) : val(v) {
        cout << "Konstruktor von " << val << endl;
    }
    ~Life() {
        cout << "Destruktor von " << val << endl;
    }
};
int main() {
    Life *l1 = new Life(1);     // Konstruktor von 1
    Life *p1, *p2;
    {
        Life *l2 = new Life(2); // Konstruktor von 2
        p1 = l2;
        Life l3(3);             // Konstruktor von 3
        p2 = &l3;
    }                           // Destruktor von 3
    delete p1;                  // Destruktor von 2
    return 0;
}
```

Welche Ausgabe erzeugt dieses Programm? Begründen Sie Ihre Antwort. Wann wird welche Ausgabe jeweils erzeugt? Markieren Sie die Stellen im Quelltext!
Lösung:
Das Programm erzeugt die folgende Ausgabe:

```
Konstruktor von 1
Konstruktor von 2
Konstruktor von 3
Destruktor von 3
Destruktor von 2
```

Konstruktor-Aufrufe erfolgen an den Stellen, wo mit `new` ein neues Objekt erzeugt wird, oder wo durch das Anlegen einer Objekt-Variable ein neues Objekt erzeugt wird. Mittels `new` angelegte Objekte werden nur zerstört, wenn es ein zugehöriges `delete` für dieses Objekt gibt – andernfalls nicht! Objekte, die zu Objektvariablen gehören, werden am Ende des Gültigkeitsbereiches der Variablen zerstört – unabhängig davon, ob es noch Zeiger auf dieses Objekt gibt. (Die Stellen, an denen eine Ausgabe erfolgt, sind im Programm durch Kommentare gekennzeichnet.)

Aufgabe 4 **(10 Punkte)**

Schreiben Sie eine C++-Template-Klasse `newVector`, die einen dynamischen Vektor mit parametrisierbaren Datentyp implementiert und eine herkömmliche Vektoraddition zweier Vektoren mit gleicher Dimension ermöglicht. Sollten die Dimensionen der Vektoren bei der Addition nicht übereinstimmen, dann soll die Additionsfunktion eine entsprechende `const char*` Execption werfen.
Sie können für die Implementierung alle C++-STL-Klassen verwenden. Auf die Angabe von `#include`-Dateien kann verzichtet werden.
Ihre Klasse sollte nahtlos in das folgende Testprogramm passen:

```
int main() {
        newVector<int> a(3), b(3), d;

        for (int i=0; i<3; i++)
                a[i] = b[i] = i;

        d = a + b;

        for (int i=0; i<3; i++)
                cout <<" "<<d[i];
        cout <<endl;
}
```

Lösung:

```
template <class T>
class newVector : public vector<T> {
public:
        newVector(int len=0) : vector<T>(len) { }
        newVector operator+(newVector &a) throw (const char*) {
                if (size()!=a.size())
                        throw "bad dim - op+ impossible!";
                newVector res(size());
                for (int i=0; i<size(); i++)
                        res[i] = operator[](i) + a[i];

                return res;
        }
};
```

Aufgabe 5 **(10 Punkte)**

Vereinfacht sehen die Logs unserer Firewall wie folgt aus:

```
134.60.67.66 134.60.66.6:33
134.60.67.31 134.60.66.6:80
134.60.67.31 134.60.66.6:80
134.60.67.31 134.60.66.1:23
```

In jeder Zeile steht ein Eintrag für eine Netzwerkverbindung mit Quelladresse und Zieladresse (samt Port). Schreiben Sie ein Programm unter **Verwendung der iostream-Bibliothek**, das die Log-Datei von der Standardeingabe liest und alle Zieladressen von Verbindungen (genau einmal) und zu jeder Zieladresse die Quelladressen (jeweils genau einmal) in Klammern ausgibt. Die Ausgabe für die obige Eingabe sollte also wie folgt aussehen:

```
134.60.66.1 (134.60.67.31)
134.60.66.6 (134.60.67.31, 134.60.67.66)
```

Tipp: Sie dürfen natürlich auch die Klassen der STL verwenden!
Lösung:

```
#include <iostream>
#include <fstream>
#include <map>
#include <string>

using namespace std;

int main() {
    map<string,map<string,int> > att;
    while (!cin.eof()) {
        char src[200];
        cin.getline(src, 200, ' ');
        if (cin.eof()) break;

        char dest[200];
        cin.getline(dest, 200, ':');
        char destport[10];
        cin.getline(destport, 10);

        att[dest][src] = 1;
    }
    for (map<string,map<string,int> >::iterator i = att.begin();
         i != att.end(); i++) {
```

```
        cout << i->first << " (";
        bool first = true;
        for (map<string,int>::iterator j = att[i->first].begin();
             j != att[i->first].end(); j++, first = false)
            cout << (first ? "" : ", ") << j->first;
        cout << ")" << endl;
    }
}
```

Aufgabe 6 **(10 Punkte)**

Gegeben ist das folgende Qt-Programm:

```
int main(int argc, char ** argv) {
    QApplication a(argc, argv);
    QWidget w;

    QGridLayout *l = new QGridLayout(&w, 2, 2);
    l->setRowStretch(0, 1);
    l->setColStretch(1, 1);

    l->addWidget(new QListBox(&w), 0, 0);

    QWidget w1(&w);
    l->addMultiCellWidget(&w1, 1, 1, 0, 1);

    QHBoxLayout *hl = new QHBoxLayout(&w1);
    hl->addStretch(1);
    hl->addWidget(new QPushButton("&Ok", &w1));
    hl->addWidget(new QPushButton("&Cancel", &w1));
    hl->addStretch(1);

    QWidget w2(&w);
    l->addWidget(&w2, 0, 1);

    QVBoxLayout *vl = new QVBoxLayout(&w2);
    vl->addWidget(new QPushButton("Hallo", &w2));
    vl->addWidget(new QPushButton("Welt", &w2));
    vl->addStretch(1);

    QWidget *w3 = new QWidget();
    w3->resize(100, 100);

    a.setMainWidget(w3);
    w3->show();
    w.show();
    return a.exec();
}
```

(a) (1 Punkt)
Wieviele Fenster zeigt dieses Programm am Bildschirm an?

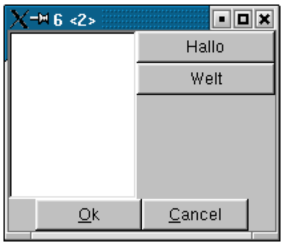
Lösung:
2 Fenster

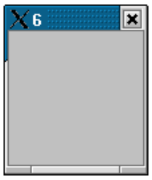
(b) (3 Punkte)
Wie verhält sich das Programm beim Schließen der Fenster/des Fensters? (Jeweils mit Angabe der Variablennamen der Fenster!) Lebt es weiter?

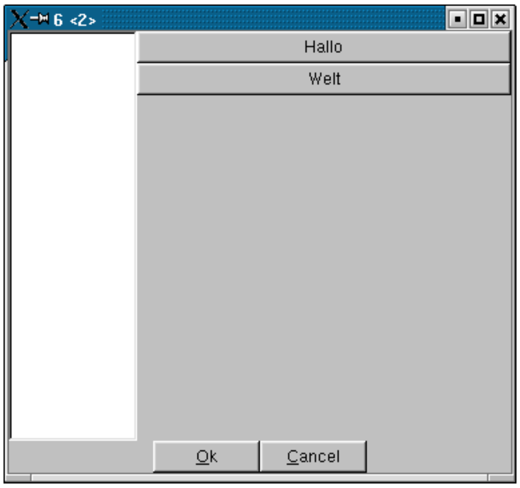
Lösung:
Beim Schliessen des Fensters mit Variable `w` (das ist das nicht-leere Fenster) bleibt `w3` und das Programm lebt weiter. Beim Schliessen des Fensters mit Variable `w3` (das ist das leere Fenster) wird das Programm beendet (und somit auch `w` geschlossen).

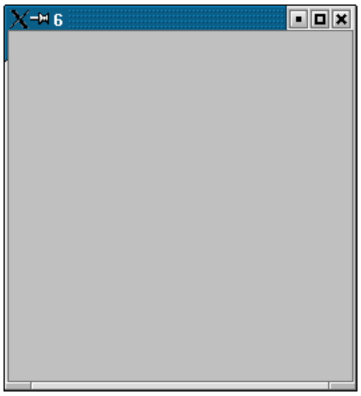
(c) (6 Punkte)
Zeichnen Sie das oder die Fenster in je zwei (deutlich) verschiedenen Größen.

Lösung:

w: 

w3: 

w: 

w3: 

Aufgabe 7 **(10 Punkte)**

(a) (5 Punkte)
Gegeben Sei die nicht vollständig implementierte Klasse K:

```
class K {
private:        int *x;         // Verweis auf Integer speichern
public:         K(int zahl);    // Zielwert uebergeben
                ~K();           // eve of destruction.
                                // something's missing here ...
};
```

Geben Sie die kompletten Köpfe (Deklarationen) aller zusätzlich explizit und implizit in K benötigten Methoden an, die bei der Verwendung von K-Objekten im folgenden Programm noch definiert werden müssen. Auf die Implementierung können Sie verzichten.

```
int main() {
        K a(1), b(1), c(2), *d = new K(4);
        // ...
        c = *d = a + b / c + 1;
        delete d;
        // das Programm soll hier noch richtig funktionieren ...
        // ... more code possible
}
```

Lösung:

```
K &operator+(const K &x);
K &operator/(const K &x);
K &operator=(const K &x);
```

oder folgende (extreme) Alternative:

```
K(const K &x);
K operator+(int x);
K operator+(K x);
K operator/(K x);
K operator=(K x);
```

oder eine korrekte (!) Zwischenlösung ;-)

(b) (5 Punkte)
Gegeben Sei die folgende Klasse F:

```
class F {
public:
        virtual ~F();
protected:
        F();
        void fleissig() {
        }
};
```

1. Welches Problem ergibt sich, wenn man die Klasse F verwenden will und wie kann man es lösen?
Lösung:
Da der Konstruktor protected ist, kann man ausserhalb von F (und seinen Unterklassen) kein F-Objekt erzeugen. Dies lässt sich dadurch umgehen, indem man eine Unterklasse von F bildet und deren Konstruktor public macht.

2. An welches andere Konstrukt aus der Vorlesung erinnert Sie das Verhalten?
Lösung: an abstrakte Klassen

3. Der Konstruktor `F()` sei jetzt nicht mehr `protected`, sondern wieder `public`. Verändern Sie die Klasse F so, dass der ursprüngliche, unter 1. beschriebene Effekt, noch einmal, aber auf eine neue Art erzielt wird:
Lösung:

```
class F {
public:
        F();
        virtual ~F();
protected:
        virtual void fleissig() = 0;
};
```