Fakultät Informatik/Mathematik

HOCHSCHULE FÜR
TECHNIK UND WIRTSCHAFT
DRESDEN
UNIVERSITY OF APPLIED SCIENCES

Lehrstuhl für Programmierung, Compilerbau/Interpreter, Mikrocontroller

Vorlesungsmitschrift

PROGRAMMIERUNG 2

Mitschrift von

Falk-Jonatan Strube

Vorlesung von

Prof. Dr.-Ing. Arnold Beck

5. Mai 2017

INHALTSVERZEICHNIS

HINWEISE

Unterlagen unter:

```
cd /home/rex/fi1/nestler/Programmierung_II_2016/
```

Compiler

- Intel i16, i13 (für Linux oder Visual Studio) www.hocomputer.de (kostenpflichtig)
- gcc 5.3, 4.85 gcc.gnu.org

Zugriff auf Windows-Programme (Visual Studio 2013) in Linux-Laboren:

```
1 rdesktop -f its56 # oder its59
```

Empfohlene Literatur: Breymann[1]

1 C++

1.1 EIN- UND AUSGABE

(siehe Folie CPP_01_stdio)

```cpp
#include <iostream> // alternativ zu <stdio.h> in C
using namespace std;  // namespace: für bestimmte Abkürzungen (bpsw. cout anstatt std::cout)
// Hinweis: "::" zeigt, dass das davorstehende "static" ist (hier: std)

class integer {}; // class: Vergleichbar mit typedef

int main() {
  integer i0; // i0: Instanz bzw. Objekt der Klasse integer
  cin.get();  // Eingabe abwarten
}
```

(vgl. integer.cpp)

```cpp
#include <iostream>
using namespace std;

class integer { // int - Variable in class verpacken
//  private:    // private ist default
    int i;  // this->i bzw. (*this).i
    // private nur für andere Klassen, andere Instanzen der gleichen Klasse können drauf zugreifen
  public: // wenn nichts steht, wird automatisch das vorherige angenommen. Hier: private
    integer(int i=0):i(i){  // Konstruktor und Defaultkonstruktor
      // i=0 default Wert, wenn keiner Angegeben
      // :i(i) übergebenes i wird dem i der Klasse zugewiesen: i_1(i_2) i_1 ist this->i, und i_2 ist das übergebene i
      cout<<"integer-Objekt i = "<<this->i<<endl;
    }

    int get(){ return i; }

    void set(int i=0){ this->i = i; }

    // statische Methode: aufrufbar ohne Instanziierung der Klasse
    static integer add(integer i1, integer i2){  // Wertkopien von i1 und i2
      // return integer(i1.i + i2.i); // alternativ und explizit: Konstruktor-Aufruf
      // return erstellt eine Kopie (mit Konstruktur erstellt)!
```


```
23        // return i1.get()+i2.get();    // Umwandlung int nach
              integer, Aufruf Konstruktor implizit
24        return i1.i + i2.i;
25        // i1.i Möglich, da innerhalb der Klasse integer und somit
              privates i sichtbar
26    }
27 };
28
29 auto max(int x, int y) -> int { return x>y ? x : y; } // Lambda-
       Funktion
30 // auto: Rückgabetyp ergibt sich aus dem Kontext bzw. über das "->
       int"
31
32 template<typename Typ1, typename Typ2>  // Weiterentwicklung Makro:
       wählt automatisch Typ aus
33 auto quotient(Typ1 a, Typ2 b) -> decltype(a/b) { return a/b; }
34
35 auto main() -> int {
36   auto k = 0;         // C++11: da 0 vom Typ int ist auch k vom Typ
         int
37   decltype(k) j = 5; // C++11: da k vom Typ int ist auch j vom Typ
         int
38   char *c = nullptr; //C++ 11: Zeigerliteral
39   int *ip = NULL;
40
41   integer i0(5), i1=4;     // 2 (alternative) Initialisiierungen
         von Objekten
42   // i1=4 nur möglich, wenn 1 Parameter gefordert ist.
43   cout<<"i0.i = "<<i0.get()<<endl;
44   // cout im Vergleich zu printf() typsicher.
45   cout<<"i0.i + i0.i = "<<integer::add(i0, i0).get()<<endl; //
         Aufruf static-Methode add
46   integer i3 = integer::add(i0, i0);                          //
         Initialisierung von i3
47   cout<<"i3.i      = "<<i3.get()<<endl;
48   i0.set(22);
49   cout<<"i0.i      = "<<i0.get()<<endl;
50   cout<<"max(3,5)  = "<<max(3,5)<<endl;
51   cout<<"5/3       = "<<quotient(5, 3)<<endl;
52   cout<<"5.0/3.0   = "<<quotient(5.0, 3.0)<<endl;
53   cout<<"b / 1     = "<<quotient('b', '1')<<endl;
54   cin.get();
55 }
```

(vgl iostream.pdf)

- nach jeder cin Eingabe: „cin.clear();“, damit Fehler ignoriert werden um weiter cin's abhandeln zu können (vgl. robust_ea)

Einlesen:

```
1 char sc;
2 cout << "sc=";
```


```
cin >> sc;
cin.clear();  // clear, um die Eingabezeile freizumachen, damit man
    nicht an Falscher Eingabe hängen bleibt
cin.ignore(INT_MAX, '\n');  // braucht #include <limits.h>
cout << "sc" << dec << (int) sc << endl;

// alternativ:
char vb[128];
cout << "s=";
cin.getline(vb, sizeof(vb), '\n');  // lesen als String, dann
   wieder umwandeln (liest auch Leerzeichen ein)
sc = atoi(vb);  // braucht #include <cstdlib>

// alternative zu getline:
cin.get(...); // lässt aber \n im Strom
cin.get();

// alternativ
cin >> setw(sizeof(vb)) >> vb;  // verhindert Überlauf
sc = atoi(vb);

// alternativ
String s; // braucht #include <string>
size_t ie=0;
cin >> s;
unsigned int ni = stoi(s, &ie, 10);

// alternativ
getline(cin, s, '\n');
double d = stod(s, &ie);

// zum compilieren: g++ p2a1.cpp -std=c++11 -o a.out
```

robust_ea1.cpp:

```
do {
  cout<<"d = "; cin>>d;   // einlesen
  if(cin.eof()) break;    // break bei Strg+D oä.
  if(cin.fail() || (cin.peek() != '\n')){   // ist nächstes Zeichen
      ungültig?
    cin.clear(); cin.ignore(INT_MAX,'\n');  // Strom zurücksetzen
       und zum \n gehen
    continue;
  }
  break; % Schleife verlassen, wenn korrekte Eingabe
} while(true);

if(cin.eof()){ cin.clear(); cout<<"eof\n"; }
else {
  cin.clear(); cin.ignore(INT_MAX,'\n');
  cout<<"Wert d = "<<d<<endl;
}
```


```
16 cin.ignore();
```

1.2 DEFAULTARGUMENTE

Defaultargumente müssen immer von rechts angefangen definiert sein:

```
1 myFunc(int i = 5, int j = 7)   // korrekt
2 myFunc(int i, int j = 7)       // korrekt
3 myFunc(int i = 5, int j)       // falsch!!!
```

1.3 ÜBERLADEN

overload.pdf

Hinweis: cast auf zwei Möglichkeiten:

```
1 int i = 5;
2 double d;
3 d = (long) i;
4 d = long(i);
```

1.4 TYPISIERTE KONSTANTEN

1.5 REFERENZEN

referenzen.pdf
Ein Speicherplatz wird mit mehreren Variablen-Namen beschrieben.

1.6 STRING0

1.7 STRING

1.8 CONST

const_mutable.pdf
Faustregel: Alle Funktionen, die nichts ändern, immer das const anfügen.

1.9 FOLGE

Initfolge/Initfolge.pdf

1.10 KOPIERKONSTRUKTUREN

Kopierkonstruktoren.pdf

1.11 FRIEND

Zugriff auf private Klassen von außerhalb:

- Funktionen
- Alle Methoden anderer Klassen
- spezifische Methoden anderer Klassen

Muss von Klasse mit privater Funktion festgelegt werden. Freundschaft wird nicht „erwidert“, ist nicht reflexiv (bsp. friendreflex).

1.12 VERERBUNG

vererben.pdf

1.13 VIRTUELLE FUNKTIONEN

- überschriebene Funktionen
- virtuelle Funktionen ⇒Späte Bindung
- rein virtuelle Funktionen
  eine Klasse mit rein virtuellen Funktionen ist abstrakt
- VMT virtual method table ABB 124

besch1.cpp (Beschäftigte)

```cpp
#include <iostream>
#include <string>
using namespace std;

class Besch{
  private:
    string name;
  public:
    Besch(string name): name(name){}  // :name(name) --> Member-Initialisierer
    string getName() {return name;}
    void setName(string name) {this->name = name;}
    virtual void display(ostream& os) {os<<name;} // virtuelle Funktion
    virtual double calc(){return 0.0;}  // normalerweise Geld nicht mit double, sondern ganzzahlig rechnen!
    // wenn Funktion eigentlich nicht benötigt wird: Polymorphes Interface:
    // virtual double calc() = 0; // rein virtuelle Funktion
    // diese rein virtuelle Funktion macht den Datentyp Besch abstract! D.h. davon kann keine Instanz erzeugt werden (wie in der Main passiert)!
};
```

```
18 ostream& operator<<(ostream& os, Besch& b) {b.display(os);}
19
20 class Arbeiter:public Besch{  // public: durchsichtige Vererbung
21   private:
22     int stunden;
23     double stdLohn;
24   public:
25     Arbeiter(string name, double stdLohn):Besch(name){ // :Besch(
          name) --> Basis-Initialisierer
26       this->stdLohn = stdLohn;
27     }
28     int getStunden() { return stunden; }
29     void setStunden(int stunden) { this->stunden = stunden; } //
          usw. getter und setter
30     void display(ostream& os){ Besch::display(); os<<""<<stdLohn<<"
           "<<stunden<<" "<<calc();}
31     double calc(){ return stdLohn * stunden; }
32 };
33 ostream& operator<<(ostream& os, Arbeiter& a) { a.display(os); }
34
35 class Angest:public Besch{
36   private:
37     double gehalt;
38   public:
39     Angest(string name, double stdLohn):Besch(name){
40       this->gehalt = gehalt;
41     }
42     void display(ostream& os){ Besch::display()<<" "; os<<gehalt<<"
           ";}
43     double calc(){ return gehalt; }
44 };
45 ostream& operator<<(ostream& os, Angest& a) { a.display(os); }
46
47 class Haendler:public Arbeiter{
48   private:
49     double prov;
50     double ums;
51   public:
52     Haendler(string name, double stdLohn, double prov):Arbeiter(
          name, stdLohn){this->prov=prov;};
53     void setUms(double ums){this->ums = ums;}
54     double calc(){return Arbeiter::calc() ++ ums*prov;}
55     void display(ostream &os){Arbeiter::display(os);}
56 }
57
58 int main() {
59   Besch* b1 = new Besch("Hans Huckebein");
60   b1->display(cout); cout<< " Euro"<<endl;
61   cout<< (*b1) << " Euro" << endl;
62   cout<< "=============="<<endl;
63   Arbeiter* b2 = new Arbeiter("Moriz Lehmann",  8.99);
```


```
64    b2->setStunden(140);
65    b2->display(cout); cout<< " Euro"<<endl;
66    cout<< (*b2) << " Euro"<< endl;
67    cout<< "==============="<<endl;
68    Angest* b3 = new Arbeiter("Friedrich Lempel",  3099.00);
69    b3->display(cout); cout<< "Euro"<<endl;
70    cout<< (*b3) << " Euro"<< endl;
71    cout<< "==============="<<endl
72    Haendler* b4 = new Handler("Bang Ohlufson",  15.80, 0.1);
73    b4->setStunden(350);
74    b4->setUms(12000);
75    b4->display(cout); cout<< "Euro"<<endl;
76    cout<< (*b3) << " Euro"<< endl;
77    cout<< "==============="<<endl
78
79    Besch* be[] = {b1,b2,b3,b4};
80    for (int i=0; i<4; i++){
81      cout<<i<<": ";
82      be[i]->display(cout); cout<<endl; // gibt 0 für alle zurück, da
            nur calc() (und display()) vom Typ Besch ausgeführt, da das
            Array aus Besch besteht! Die Funktion calc() wurde ü
          berschrieben (Vgl. Überladen, wenn bei gleichen Namen
          verschiedene Paramater angegeben sind)!
83      // Pointer im Array bestimmt die ausgeführten Funktionen!
84      // Lösung: "virtual" als Schlüsselwort vor Funktion (s.o.).
          Angeben bei erstem Auftauchen in Vererbungshierarchie
85      cout<<(*(be[i]))<<endl<<"+++++++++++++++"<<endl;  // geht auch,
            da im Operator (die ja nicht virtuell gemacht werden kann,
          weil sie kein Member ist) die virtuelle display-Fkt
          aufgerufen wird.
86    }
87
88    return 0;
89  }
```

Vererbungs Hierarchie:
ABB 125
Potentielle Mehrfachvererbung (bei Manager):

- Hat mehrfache Namen (wegen unterschiedlichen Vererbungslinien)

Problemlösung: Arbeiter, Angest und Händler von virtual Besch erben lassen (dort, wo es sich aufzweigt):

```
1 class Arbeiter: virtual public Besch{};
```

Hat immer noch Einschränkungen!

1.14 TEMPLATES

Templates Folie (enthält auch exception). Vor allem für Containerklassen/Collections verwendet (Vektoren, Hashtables, Listen).
Schafft Möglichkeit Klassen parameterbehaftet zu generieren (generische Klassen in Java).

Funktionen von Templates müssen (im Gegensatz zur bisherigen Vorgehensweise) im header-File ausprogrammiert werden. Alles, was zum Template gehört kommt ins header-file!!!

Beispiel Array Intarr.h

```
const int SizeArr = 24;

// Konstruktor, Kopierkonstruktor und Destruktor werden bei Klassen
    mit Pointer immer benötigt!
class IntArr{
  public:
    IntArr (int Gr = SizeArr )  // Konstruktor
    IntArr (const IntArr&)      // Kopierkonstruktor
    ~IntArr() (delete IA;}      // Destruktor
    IntArr & operator = (const IntArr&);
    int& operator [] (int i);
    int getNum() const {return Size;}
    void resize(int NewSz);
  private:
    int Size;
    int * IA;
};
```

Auszug Intarr.cpp

```
...
IntArr :: IntArr (int Sz){
  Size = Sz;
  IA = new int [Size];
  for (int i=0; i<Sz; ++i) IA[i] = 0;
}
```

Modifikation um es generisch zu machen (Template), Inhalte von Intarr.cpp müssen eingefügt werden:
Intarr.h

```
const int SizeArr = 24;

template <class T>
class IntArr{
  public:
    IntArr (int Gr = SizeArr );
    IntArr (const IntArr<T>&);
    ~IntArr() (delete IA;}
    IntArr & operator = (const IntArr<T>&);
    T& operator [] (int i);
    int getNum() const {return Size;}
    void resize(int NewSz);
  private:
    int Size;
    T * IA;
};

```

```
//... + Inhalt von Intarr.cpp!!!
template <class T>
IntArr<T> :: IntArr (int Sz){
  Size = Sz;
  IA = new T [Size];
  for (int i=0; i<Sz; ++i) IA[i] = 0;
}

template <class T>
IntArr<T> :: IntArr (const IntArr<T>& Other){
  Size = Other.Size;
  IA = new T[Size];
  for (int i=0; i<Size; i++) IA[i]=Other.IA[i];
}
usw.
```

in der Main dann:

```
// vorher: IntArr IA (10);
IntArr<int> IA (10);
// oder auch
IntArr<double> IA (10);
// auch möglich
IntArr<Fraction> IA (10); // wobei Fraction in einer anderen Klasse
    definiert ist.
```

Beispiel Listen cobject.h

2 JAVA

2.1 GRUNDLAGEN

Java0intro.pdf
Klassenname und Dateiname müssen exakt übereinstimmen. Jede Datei darf nur eine Klasse enthalten.

```
public class HelloOne{  // Dateiname: HelloOne.java
  public static void main(String args[]}{ // im Vergleich zu C++
     gibt es keinen public-Bereich, die Eigenschaft muss vor jede
     Funktion/Variabel geschrieben werden.
    System.out.println("welcome to java");
    System.out.println("today is " + new java.util.Date());
    System.out.printf("JAVA test: %d\n", 26);
  }
}
```

javac HelloOne.java zum Übersetzen
java HelloOne zum Ausführen

```
public class HelloEcho{
  public static void main(String args{]){
    for (int i = 0; i<args.length; i++)
      System.out.println(args[i]);
    // oder:
    for (String s:args)
      System.out.println(s);
    System.exit(0);
  }
}
```

2.1.1 PARSEN VON DATEN

```
public class HelloParse{
  public static void main(String args{]){
    int i;
    double d;
    i = Integer.parseInt(args[0]);
    d = Double.parseDouble(args[1});
    System.out.println("i: "+i+" d: "+d);
  }
}
```

2.1.2 GUI MIT AWT

```java
import java.awt.*;
import java.awt.event.*;
import java.util.*;
public class HelloTwo extends Panel { // extends: erweitert. Also: "erbt von"
  Label l1=new label("welcome to java", Label.CENTER);
  Label l2=new Label("today is "+new Date(), Label.CENTER); // wegen import javo.util.* kann hier nur Date() geschrieben werden.
  Button close = new Button ("Close");

  public HellowTwo(){
    setFont(new Font("System", Font.PLAIN, 24));
    setLayout(new GridLayout(3,1,0,10));
    add(l1);  // alternativ: this.add(l1);
    add(l2);
    close.addActionListener(new ActionListener(){
      public void actionPerformed(ActionEvent e){
        // bpsw erweiterbar mit System.out.printl("Close...");
        System.exit(0);
      }
    });
  }

  public static void main(String args[]){
    Frame f=new Frame("first Desktop Application");
    HelloTwo h = new HelloTwo();
    f.add(h);
    f.setVisible(true);
    f.pack();
  }
}
```

2.1.3 APPLET

```java
import java.applet.*;
import java.awt.*;

public class HelloThree extends Applet{
  public void init(){
    Panel h= new HelloTwo();
    add(h);
  }
}
```

```html
<html>
<body>
<applet code=HelloThree width=940 height=540></applet>
</body>
</html>
```


Zu jedem Applet gehört ein dazugehöriges HTML-File. Damit kann das Applet über einen Applet-Viewer benutzt werden. Hier funktioniert der Close-Button aber nicht.

Angepasstes HelloTwo Hiermit wird der Close-Button implementiert, sodass er im Applet nicht erscheint.

```
// Quit-Button wird erst in main-Funktion eingeführt
```

java1.pdf

2.1.4 OPERATOREN

- $>>$ schiebt nach rechts und fügt abhängig vom ersten Bit (Vorzeichen) 0en (1. Bit 0) oder 1en (1. Bit 1) nach. $>>>$ schiebt immer 0en nach.

2.1.5 EINGABEKONVERTIERUNG, FORMATIERTE AUSGABE / KONVERTIERUNG NUMERISCHER WERTE

```
public class Num{
  public static vaid main(String args[]){
    int x;
    x = Integer.parseInt(args[0]);  -- Exceptions bei keiner
       Eingabe und bspw. String-Eingabe
    System.out.printf("x: %d", x);
    System.out.println("x: "+x);
  }
}
```

2.1.6 VERGLEICH VON OBJEKTEN

- == prüft Objekt gleich ist (nicht Inhalt!)
- .equals() prüft Gleichheit von Objektinhalt

```
public class Cmp{
  public static vaid main(String args[]){
    String s="Hans";
    if (args[0] == s)
      System.out.println("args[0]==s liefert true");
    else
      System.out.println("args[0]==s liefert false");
    if (s.equals(args[0]))
      System.out.println("s.equals(args[0]) liefert true");
    else
      System.out.println("s.equals(args[0]) liefert false");
    // ergibt das erste Mal false (weil Objekte ungleich) und das
       zweite Mal true (weil Objektinhalt gleich).

    if ("Hans" == s)
```

```
15        System.out.println("Hans==s liefert true");
16      else
17        System.out.println("Hans==s liefert false");
18      if (s.equals("Hans"))
19        System.out.println("s.equals(\"Hans\") liefert true");
20      else
21        System.out.println("s.equals(\"Hans\") liefert false");
22      // ergibt beides mal true, weil der Compiler optimiert und der
           "Hans" String nicht doppelt erzeugt wurde.
23    }
24  }
```

2.1.7 ARRAYS

```
1  int intArray[]; // [] bleibt hier immer leer!
2  intArray = new int[2] // jetzt [] füllen
3  // alternativ, inkl. Initialisierung:
4  int intArray[] = {1,2,3};
5
6  // mehrdimensional:
7  int mda[][] = { {1,2}, {1,2,3}, {1,2,3,4,5} };
8  // alternativ:
9  int mda[][];
10 mda = new int{2][];
11 mda[0] = new int[2];
12 mda[1] = new in[5];
13 // oder auch:
14 mda = new int[2]{10];
```

Vgl. Byte Array (zum Verschicken über Netzwerk: Serialisierung)

2.1.8 STATEMENTS

if- und Schleifenbedingung braucht unbedingt boolsche Werte.
Also bspw. nicht ïf(i)", sondern ïf(i!=0)".

```
1  public class Cmp{
2    public static void main(String args[]){
3      for ( String x:args)  // for-each alternativ zu for(int i=0; i<
          args.length; i++) {x=args[i]; System.out.println(x);}
4        System.out.println(x);
5    }
6  }
```

2.1.9 EXCEPTIONS

```
1  class tryDemo{
2    ...
3  }
```


2.2 KLASSEN

Java2_classes.pdf

```
class LocalVar{
  public static void main(String args[]){
    int i,j;
    j=i+1;  // ergibt Fehler, weil i noch nicht initialisiert wurde
        .
    System.out.println(j);
  }
}
```

2.2.1 INITIALISIERER

Funktion ohne Funktionskopf. Wird ausgeführt, wenn Instanz erstellt wird. Ein static-Initialisierer wird ausgeführt, sobald die Klasse geladen wird.

2.2.2 VERERBUNG

Überschriebene Methoden in Java verhalten sich wie virtuelle Methoden in C++.

```
import java.awt.*;
import java.awt.event.*;

public class myPanel extends Component implements WindowListener{
  private String myString;

  public void windowActivated(WindowEvent e){}
  public void windowClosed(WindowEvent e){}
  public void windowClosing(WindowEvent e){System.exit(1);}
  public void windowDeactivated(WindowEvent e){}
  public void windowDeiconified(WindowEvent e){}
  public void windowIconified(WindowEvent e){}
  public void windowOpened(WindowEvent e){}

  @Override
  public Dimension getPreferredSize() {
    return new Dimension(400,200);
  }

  myPanel(String s){
    myString =s;
  }

  public static void main(String args[]){
    Frame f=new Frame (args.length>0?args[0]:"no String - no fun");
    myPanel p=new myPanel(args.length>0?args[0]:"no String - no fun
        ");
    System.out.println(p.myString+" in main");
    f.add(p);
    f.addWindowListener(p);
```


```
30      f.pack(); //setSize(400,200);
31      f.setVisible(true);
32      f.repaint();
33    }
34
35    @Override
36    public void paint(Graphics gc){
37      //System.out.println(myString+" in paint");
38      gc.drawString(myString,10,20);
39    }
40  }
```

2.3 ABSTRACT WINDOWING TOOLKIT (AWT)

java4_awt.pdf

2.3.1 BORDERLAYOUT

Sollen mehrere Buttons in einen Bereich (bspw. NORTH), muss ein extra Panel dort eingefügt werden:

```
1 ...
2 Panel pTop = new Panel(new FlowLayout());
3 poben.add(b1);
4 poben.add(b2);
5 add(poben,BorderLayout.NORTH);
6 ...
```

2.3.2 CARDLAYOUT

Mehrere Karten, man sieht immer nur eine. Wechseln mit next, previous oder show.

Mehrere Buttons usw. hinzufügen

```
1 String[] myLabels = {"Max", "Moritz", "Bolte"};
2 ...
3 Panel p=new Pannel(new FlowLayout());
4 for (int i=0; i<myLabels.length; i++){
5   Button b = new Button (myLabels[i]);
6   p.add(b);
7 }
8 this.add(p.BorderLayout.CENTER);
```

2.4 EVENTHANDLING

java4_eventhandlig.pdf

2.4.1 INNER CLASSES

4 Arten merken!

2.5 JAVA I/O

java3_io.pdf

2.5.1 HEXDUMP

3 PRÜFUNGSINHALT

C++: manipulator, cout, «, Ausgabestring formatieren, Operatorüberladung, Vererbung, Überladen/überschriebene/virtuelle Funktionen// Java bis einschließlich AWT

LITERATUR

[1] Ulrich Breymann und Ulrich Breymann. „Der C++ Programmierer“. In: C++ LERNEN, PROFESSIONELL ANWENDEN, LÖSUN (2009).