



Technische  
Universität  
Braunschweig



# **Grundlagen der automatischen Informationsverarbeitung für den Maschinenbau / Programmieren**

Saalübung 5 – Einführung in die Objekt-Orientierte Programmierung

# Saalübung 5

- Wiederholung
- Objektorientierte Programmierung
  - Klassen und Objekte
- Lesen und Schreiben von Dateien

## Literatur:

- Übungsskript Kapitel 7
- C++-Lehrbücher
- Internetressourcen (z.B. [www.cppreference.com/](http://www.cppreference.com/) ; [www.cplusplus.com/doc/tutorial/files/](http://www.cplusplus.com/doc/tutorial/files/) )

# Saalübung 5

- Wiederholung
- Objektorientierte Programmierung
  - Klassen und Objekte
- Lesen und Schreiben von Dateien

# Wiederholung letzte Woche (Saalübung 4 - Funktionen)

- Erledigen eine abgeschlossene Aufgabe beliebiger Komplexität
- Verarbeiten eine **variable Anzahl** übergebener **Parameter** zu **einem Ergebnis**
- Besitzen einen **Rückgabetyt** und einen **Rückgabewert** (Rückgabewert muss denselben Datentyp wie der definierte Rückgabetyt aufweisen!)
- Müssen lediglich einmal definiert werden und sind dann **beliebig oft** im Programm **verwendbar** (Verwendung erfolgt durch Funktionsaufruf)

**Prototyp:** *Rückgabetyt Funktionsname (Parameterliste);*

z.B. `double produkt(double a, double b);`

**Definition:** *Rückgabetyt Funktionsname (Parameterliste){Anweisungsblock;}*

z.B. `double produkt(double a, double b) {return a*b;}`

**Aufruf:** *Funktionsname (Aktualparameterliste)*

`produkt(5.22, 4.634) oder produkt(wert1, wert2)`

# Beispiel

1 //Funktionsprototyp/Deklaration

2 double produkt (double a, double b); Bei Funktionsdeklaration

3 int produkt (int,int); Benennung der Parameter optional

4 //Main-Funktion

5 int main(void) Schlüsselwort void ist hier  
6 { optional. Alternativ ()

7 //Funktionsaufruf

8 int ergebnis= 4;

9 ergebnis = produkt(3,ergebnis); Bei Funktionsaufruf werden Werte  
10 produkt(2,1); als Parameter übergeben. Diese  
11 } können in Variablen gespeichert  
12 sein. Ein Ergebnis kann, muss  
13 aber nicht verwendet werden

14 //Funktionsdefinition

15 double produkt(double a, double b)

16 {

17 ...

18 }

19 int produkt (int a, int y) Bei Funktionsdefinition müssen  
20 { Parameter benannt werden.  
21 ... Innerhalb der Funktion sind sie  
lokale Variablen, die mit den  
übergebenen Werten initialisiert  
sind!

22 {

23 ...

24 }

# Rekursion

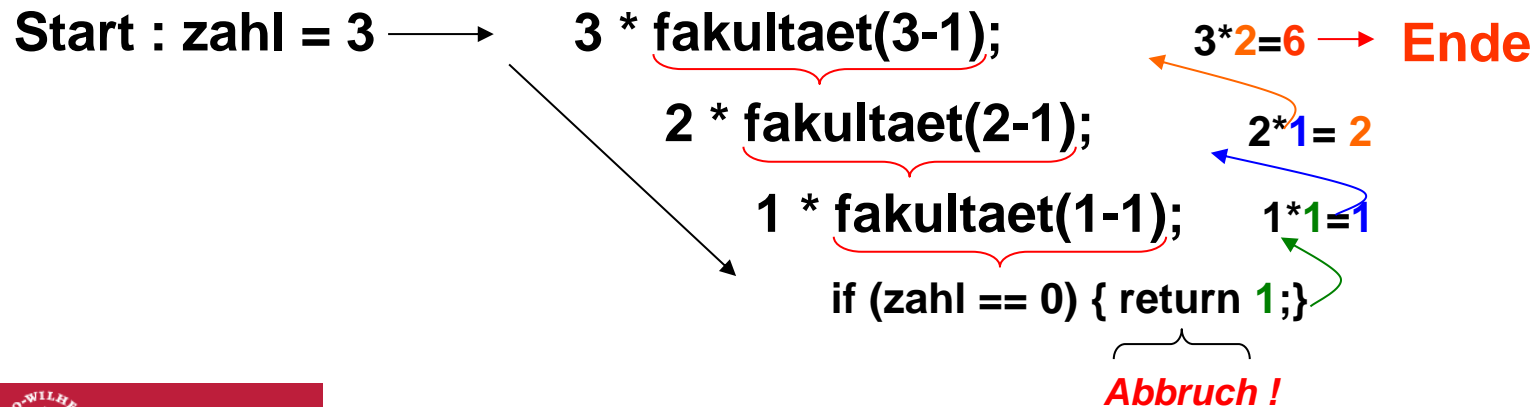
- Funktionen können sich selbst aufrufen
- Einfache Beschreibung vieler Probleme z.B. Fakultätsberechnung (vgl. Vorlesung)

```
1  int fakultaet (int zahl)
2  { if (zahl == 0) { return 1;}           // Abbruchkriterium: 0! = 1
3  else { return zahl * fakultaet(zahl-1);}
4  }
```

**Selbstaufruf** (orange arrow pointing to `fakultaet(zahl-1)`)

**Nicht zwingend nötig** (orange arrow pointing to `else`)

- Wichtig: Jede rekursive Funktion benötigt eine Abbruchbedingung!

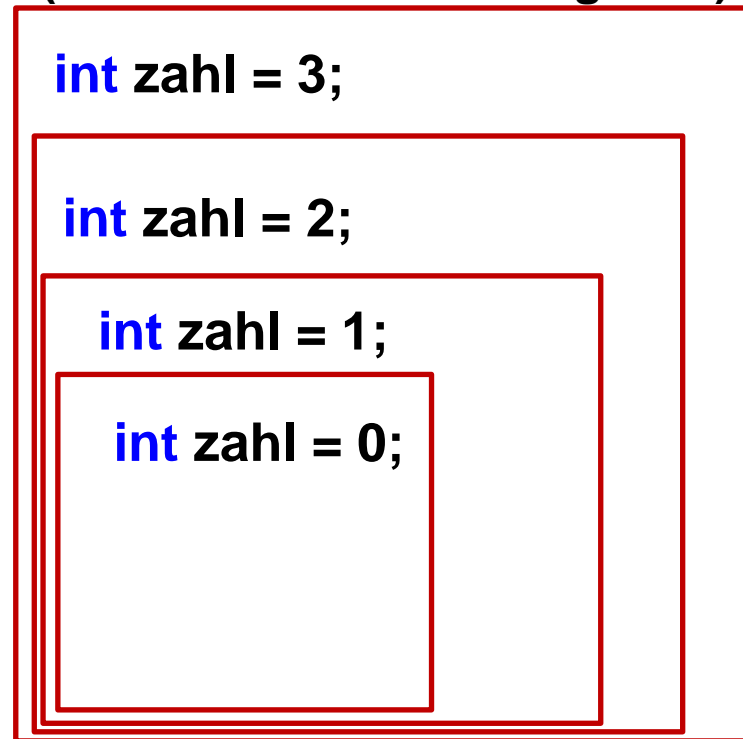
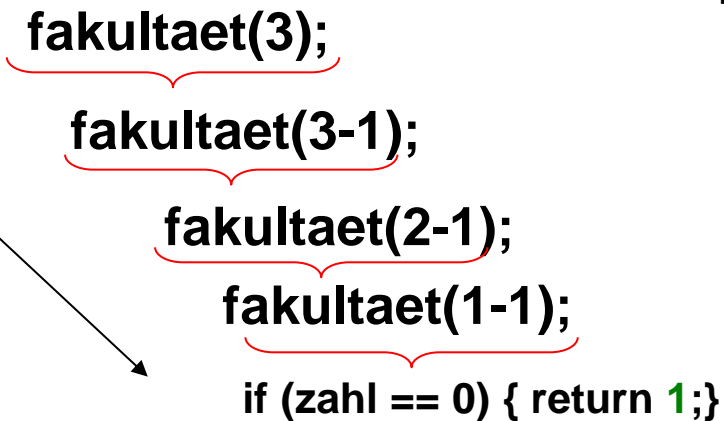


# Problem

- Rekursionen sind oft langsamer und speicheraufwändiger als iterative Berechnungen

## Benötigter Speicher für lokale Variablen (technisch evtl. anders gelöst)

**fakultaet(3);**  
    **fakultaet(3-1);**  
        **fakultaet(2-1);**  
            **fakultaet(1-1);**  
                if (zahl == 0) { return 1; }



- Bei jedem Funktionsaufruf wird zusätzlicher Speicher verbraucht, abhängig der Rechnerarchitektur

# Saalübung 5

- Wiederholung
- Objektorientierte Programmierung
  - Klassen und Objekte
- Lesen und Schreiben von Dateien



# Objekt-Orientierte(OO) Programmierung

- Motivation:

- Wiederverwendbarkeit des Quelltext (Vermeidung von Redundanz/Debugging)
- Modularisierung → Arbeitsteilung bei großen Softwareprojekten

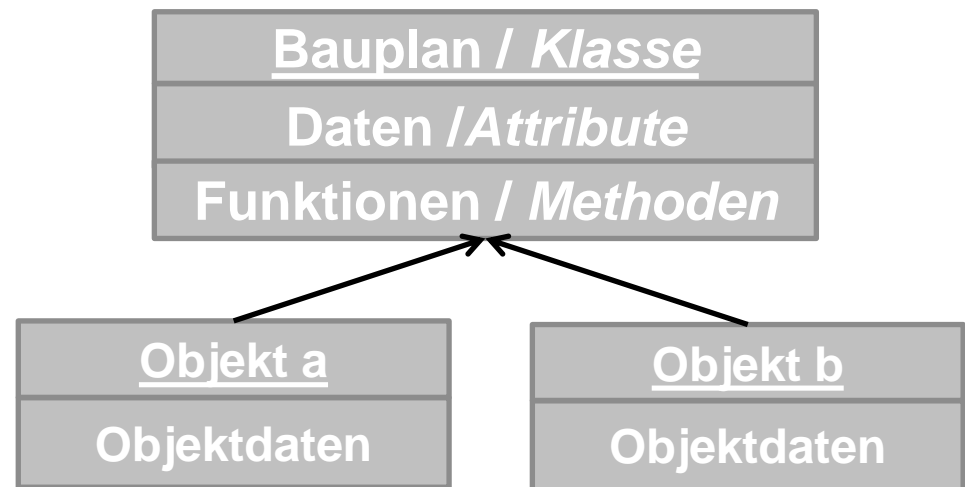
Beispiel WinXP: 45 Millionen Quelltextzeilen (*lines of code*)

- Idee: Zusammenfassung der Daten mit den darauf zugreifenden Funktionen

## Prozedural:



## Objekt-Orientiert:



- Anwendungsgebiet u.a.:

- Benutzeroberflächen (*graphical user interface GUI*) z.B. Qt, wxWidgets, Tkinter, etc.

# Begriffe in der Objekt-Orientierten (OO) Programmierung

- Klasse: Selbst generierter Datentyp / Bauplan von Objekten
- Objekt: Ausprägung / Instanz einer Klasse
- Attribute: Daten eines Objekts
- Methoden: Funktionen eines Objektes

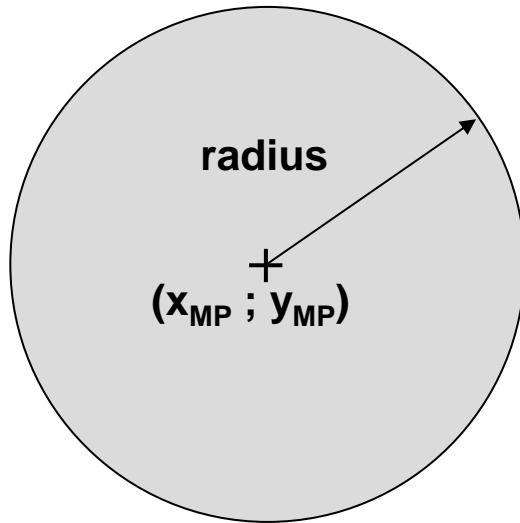
## Was muss implementiert werden?

- Klassendefinition: Bauplan der Klasse mit Attributen und Methodenprototypen
- Methodendefinition: Was passiert, wenn eine Methode aufgerufen wird

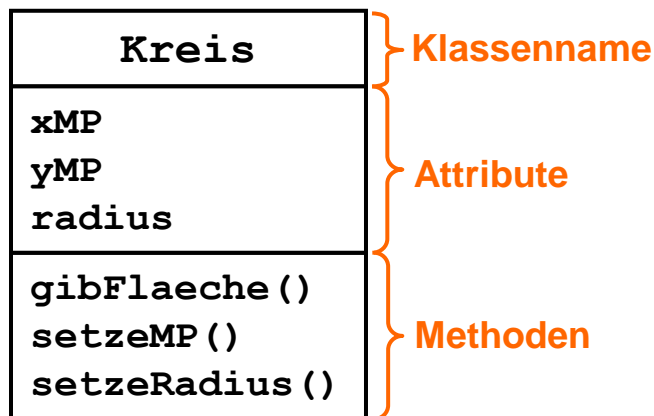
## Hauptfunktion `int main()`:

- Erstellung / Instanziierung von Objekten: `Kreis a;`
- Aufruf von Methoden: `a.setzeMP(1.0, 2.0);`

# Beispiel einer Klasse anhand der Beschreibung eines Kreises



UML-Symbol der Klasse Kreis



- Zur Beschreibung eines Kreises im 2D-Raum werden folgende Daten benötigt:

- Kreismittelpunktskoordinaten  $x_{MP}$ ,  $y_{MP}$
- Kreisradius `radius`

→ Die zu speichernden Daten `xMP`, `yMP` und `radius` werden **Attribute** genannt.

- *Methoden*, die bei einem Kreis denkbar sind:

- Berechnung der Fläche  
`double gibFlaeche (void)`
- Setzen des Mittelpunkts (Mittelpunkt festlegen)  
`void setzeMP (double x, double y)`
- Setzen des Radius (Radius festlegen)  
`void setzeRadius (double r)`

→ Zu einer Klasse gehörige Funktionen `gibFlaeche`, `setzeMP`, `setzeRadius` werden **Methoden** genannt.

# Klassendefinition – Syntax

- Beschreibt die Struktur / den Bauplan einer Klasse
- Klassendefinition wird mit **class** eingeleitet und endet mit **;**
- Enthält die Definition der Attribute und die Prototypen der Methoden

```
1 // Allgemeine Klassendefinition
2 class Klassenname {
3     Datentyp attribut1;           // Attributdefinitionen
4     Datentyp attribut2;
5     ...
6
7     Rückgabetyt methode1 (Parameterliste); // Methodenprototypen
8     Rückgabetyt methode2 (Parameterliste);
9     ...
10 };
```

Hinweis: Implementierung der Methoden (vgl. Funktionsdefinition)  
darf in die Klassendefinition mit aufgenommen werden,  
sollte aber aus Gründen der Übersichtlichkeit vermieden werden.

# Beispiel: Klassendefinition Kreis

```
1  #include <iostream>                // Präprozessor-Anweisungen
2  using namespace std;
3
4  class Kreis
5  {
6      double xMP;
7      double yMP;
8      double radius;
9
10     double gibFlaeche(void);
11     void setzeMP(double xMP, double yMP);
12     void setzeRadius(double r);
13 };
```

Attributdefinitionen

Methodenprototypen

**Wird in der Prüfung sehr gerne vergessen!**

# Methodendefinition – Syntax

- In der Klassendefinition sind lediglich die Prototypen der Methoden enthalten:  
`Rückgabetyp method1(Parameterliste);`
- Definition der Methoden erfolgt außerhalb der Klassendefinition:

```
1  // Methodendefinition
2  Rückgabetyp Klassenname::method1(Parameter) {
3
4      int i = 2;                // Initialisierung lokaler Variablen
5      double wert;              // Definition lokaler Variablen
6
7      wert = i * Parameter;      // Methodenname
8      attribut1 = wert;          // Zugriff auf Attribut
9      Methodenanweisung;        // weitere Anweisungen
10     ...
11
12     return Rückgabewert;
13 }
```

**Klassenname + scope-Operator (::)**  
**zeigen dem Compiler, zu welcher Klasse**  
**die Methode gehört.**

# Beispiel: Methodendefinition für Klasse Kreis

```
1  // Methodendefinitionen
2  #include <iostream>
3  using namespace std;
4
5  double Kreis::gibFlaeche(void) // Methode gibFlaeche
6  {
7      return 3.14159*radius*radius;
8  }
9
10 void Kreis::setzeMP(double xMP, double yMP) // Methode setzeMP
11 {
12     this->xMP = xMP;
13     this->yMP = yMP;
14 }
15
16 void Kreis::setzeRadius(double r) // Methode setzeRadius
17 {
18     radius = r; // alternativ: this->radius = r;
19     cout << "Radius wurde gesetzt auf: " << r << endl;
20 }
```

**this->** steht für die Adresse des zugehörigen Objekts → Unterscheidung zwischen lokalen Methodenparametern und Klassenattributen

# Was ist ein Objekt?

- Eine Klasse definiert die Struktur eines komplexen Datentyps.
- In der objektorientierten Programmierung werden diese „*Klassen-Variablen*“ als **Objekt** oder **Instanz** bezeichnet.
- Der Vorgang der Definition wird dementsprechend als **Instanziierung eines Objektes** der jeweiligen Klasse bezeichnet.
- Jedes **Objekt** fordert Speicher für seine **Attribute** an.
- **Objekte** ein und derselben Klasse unterscheiden sich also durch die Werte ihrer **Attribute**.
  - Jedes Objekt einer Klasse erhält eigenen Speicher für seine Attribute.
- **Objekte** „teilen“ sich die **Methoden** ihrer Klasse, denn *Methoden operieren lediglich auf Daten und speichern keine Daten*.
  - Methoden verschiedener Objekte derselben Klasse sind identisch.



# Instanziierung von Objekten und Zugriff auf Objekt-Elemente

Instanziierung eines Objektes erfolgt analog zur Definition einer Variablen:

```
1 // Allgemeine Klassendefinition
2 class Klassenname {
3     Datentyp attribut1;           // Attributdefinitionen
4     ...
5     Rückgabetyyp methode1(Parameterliste); // Methodenprototypen
6     ...
7 };
8
9 // Instanziierung eines Objektes
10 // (vgl. Variablendefinition z.B. int zahl1)
11
12 ...
13 int main() {
14     ...
15     Klassenname objektname;      // Objekt Instanziierung
16
17     objektname.attribut1;         // Zugriff auf Objekt-Attribut
18
19     objektname.methode1(Parameterliste); // Zugriff auf Objekt-Methode
20     ...
21     return 0;
22 }
```

**Zugriff auf Elemente (Attribute und Methoden) des instanziierten Objekts objektname mittels „Punkt“-Operator (.)**

# Datenkapselung

- Zentrales Prinzip der objektorientierten Programmierung zur Abschirmung von Elementen auf Zugriffe von „außen“.
- Folgende Möglichkeiten bestehen bei der Definition oder Deklaration der Klassenelemente:
  - **private:**
    - Zugriff nur innerhalb der Klasse durch Elemente (Methoden) derselben Klasse möglich („Zugriff von innen“).
    - Direkter Zugriff aus anderen Funktionen nicht möglich.
    - Verwendung für Attribute
  - **public:**
    - Zugriff von Außen ist möglich („Zugriff von außen“).
    - Verwendung für Methoden

Zugriff auf die **privaten Attribute** einer Klasse erfolgt also über die **öffentlichen Methoden** der Klasse, wodurch ungewollte bzw. fehlerhafte Veränderungen der Werte der Attribute ausgeschlossen werden können.

# Beispiel 5-1: Anwendung einer Klasse Kreis

```
1 // bsp5-1.cpp: Anwendung einer Klasse Kreis
2 #include <iostream> // Präprozessor-Anweisungen
3 using namespace std;
4
5 class Kreis {
6     private : // Attributdefinitionen
7         double xMP;
8         double yMP;
9         double radius;
10    public: // Methodenprototypen
11        double gibFlaeche(void);
12        void setzeMP(double xMP, double yMP);
13        void setzeRadius(double r);
14    };
15 // Methodendefinitionen
16 double Kreis::gibFlaeche(void) { // Methode gibFlaeche
17     return 3.14159*radius*radius;
18 }
19 void Kreis::setzeMP(double xMP, double yMP) { // Methode setzeMP
20     this->xMP = xMP;
21     this->yMP = yMP;
22 }
```

**Datenkapselung**

**Methodendefinition mittels scope-Operator :: (vgl. Folie 11f)**

**kein Konflikt, da Unterscheidung durch this-> Zeiger gegeben (vgl. Folie 12)**

# Beispiel 5-1: Anwendung einer Klasse Kreis

```
23 void Kreis::setzeRadius(double r) { // Methode setzeRadius
24     radius = r; // alternativ: this->radius = r;
25     cout << "Radius wurde gesetzt auf: " << r << endl;
26 }
27
28 // Hauptprogramm
29 int main() {
30
31     Kreis kreis1;
32     Kreis kreis2;
33
34     kreis1.setzeMP(0.0,0.0);
35     kreis2.setzeMP(2.0,3.0);
36
37     kreis1.setzeRadius(1.0);
38     kreis2.setzeRadius(2.0);
39
40     cout << "Flaeche kreis1: " << kreis1.gibFlaeche() << endl;
41     cout << "Flaeche kreis2: " << kreis2.gibFlaeche() << endl;
42
43     return 0;
44 }
```

**Instanziierung der Objekte kreis1 und kreis 2**

**Zugriff auf Elemente (hier Methoden) der instanzierten Objekte mittels „Punkt“-Operator ( . )**

**Ausgabe: Fläche kreis1: 3.14159  
Fläche kreis2: 12.5664**

# Vergleich der prozeduralen mit der OO-Programmierung

Beispiel: Hauptprogramme für Kreisberechnung:

```
// OO-Programm
int main(){
    ...
    Kreis a;
    a.setzeMP(1.0,2.0);
    a.setzeRadius(3.0);
    cout<< a.gibFlaeche();
    ...
    return 0;
}
```

```
// Prozedural
int main(){
    ...
    double a_MP_x=1.0;
    double a_MP_y=2.0;
    double a_radius=3.0;
    cout<<
    berechneKreisflaeche(a_MP_x,a_MP_y,a_radius);
    ...
    return 0;
}
```

Vorteile bei Objekt-Orientierten Programmierung:

- Kapselung der Daten:
  - Zusammengehörige Daten in einem Objekt zusammengefasst
  - Kontrolle darüber, dass Daten nur sinnvoll verändert werden können
- Erhöhte Lesbarkeit des Quelltextes

# Saalübung 5

- Wiederholung
- Objektorientierte Programmierung
  - Klassen und Objekte
- Lesen und Schreiben von Dateien

# Lesen und Schreiben von Dateien

Problem:

Nach dem Durchlauf des Programms → Löschen aller Variablen und Ergebnisse

Abhilfe:

Dauerhaftes Speichern durch Ablegen in einer Datei auf der Festplatte

Anmerkung:

- Es gibt viele verschiedene Bibliotheken zum Schreiben und Lesen von Dateien.
- Daten können im lesbaren (ASCII-)Format oder im binären Format gespeichert werden.
- Es gibt verschiedene Standard-Formate zum Abspeichern von Daten (z.B. hdf5, netcdf, xml, csv etc.).

**Im Folgenden wird das Schreiben im ASCII-Format mit Hilfe der *fstream*-Bibliothek gezeigt.**

Weitere Infos unter [www.cplusplus.com/doc/tutorial/files/](http://www.cplusplus.com/doc/tutorial/files/)

## Beispiel 5-2: Schreiben von Dateien

Die verwendete `fstream`-Bibliothek ist von der bekannten `iostream`-Bibliothek abgeleitet.

→ Die Nutzung ist nahezu identisch!

```
1  #include <iostream>
2  #include <fstream>
3  using namespace std;
4
5  int main () {
6
7      float a=1.56;
8      ofstream myfile;
9      myfile.open("example.txt");
10     myfile << "Writing this to a file.\n";
11     myfile << "a : " << a << endl;
12     myfile << "End of File";
13     myfile.close();
14
15     return 0;
16 }
```

**Einbinden der `fstream`-Bibliothek**

**Instanziierung eines `ofstream`-Objekts**

**Erstellen und Öffnen der Datei `example.txt` mit Hilfe der Methode `open(char*)`**

**Schreiben in die Datei `example.txt` (analog zu `cout`)**

**Schließen der Datei `example.txt` mit Hilfe der Methode `close()`**



# Beispiel 5-3: Lesen von Dateien

Beim Lesen wird zusätzlich auf die `string`-Bibliothek zurückgegriffen!

```
1  #include <iostream>
2  #include <fstream>
3  #include <string>
4  using namespace std;
5
6  int main () {
7
8      string line;
9      ifstream myfile;
10     myfile.open("example.txt");
11     while ( myfile.good() )
12     {
13         getline (myfile,line);
14         cout << line << endl;
15     }
16     myfile.close();
17
18     return 0;
19 }
```

Einbinden der `fstream`- und `string`-Bibliothek

Instanziierung eines `string`-Objekts

Instanziierung eines `ifstream`-Objekts

Öffnen der Datei `example.txt` mit Hilfe der Methode `open(char*)`

Lesen und Ausgeben der Zeilen

Schließen der Datei `example.txt` mit Hilfe der Methode `close()`

# Zusammenfassung

## Begriffe in Saalübung 5:

- Klassen
- Klassendefinitionen - Syntax
- Attribute
- Methoden
- Methodendefinitionen - Syntax
- Objekte, Instanzen
- Datenkapselung

## Weiteres Thema:

- Lesen und Schreiben von Dateien