

Exercise Week 12

GianAndrea Müller
`mailto:muellegi@student.ethz`

May 30, 2018

Time Schedule

- 10' Wiederholung Klassen
- 10' Vererbung
- 10' Polymorphie und Abstrakte Basisklassen
- 10' Konstruktoren
- 10' Has a - is a

Learning Objectives

- Verständnis: Vererbung
- Anwendung: Polymorphismus

Klassen: Wiederholung

Was wir gelernt haben:

- Mitgliedsfunktion
- Mitgliedsvariable
- Zugriffsoperatoren

```
1  class example {
2  public:
3      void print() {std::cout<<this->member;}
4  private:
5      int member = 3;
6  };
7
8  int main(){
9      example a;
10     a.print();
11 }
```

Klassen: Wiederholung

Was wir gelernt haben:

- **Zugriffsbereiche**
- **Zugriffsmethoden**

```
1 class example {  
2     private:  
3         int variable;  
4     public:  
5         int get_variable();  
6         void set_variable(int value);  
7 };
```

Klassen: Wiederholung

Was wir gelernt haben:

- **Konstruktor**
- **Kopierkonstruktor**
- **Zuweisungsoperator**
- **Destruktor**

Regel der Drei

Klassen: Wiederholung

Was wir gelernt haben:

- **Operatoren**

Erkenntnisse

- Für neue Datenstrukturen brauchen wir neue Operatorüberladungen.
- Operatoren als Mitgliedsfunktionen
- Operatoren als globale Überladungen
- Implizite Typenumwandlung

Vererbung: Grundlagen

```
1  class A{
2      ... //Basisklasse
3  };
4
5  class B: public A{
6      ... //Abgeleitete Klasse
7  };
8
9  class C: public B{
10     ...
11  };
```


Vererbung: Zugriffskontrolle

Vererbung \ Mitglied	public	protected	private
public	public	protected	n/a
protected	protected	protected	n/a
private	private	private	n/a

Polymorphismus

```
1  class A {
2      virtual void print()
3      {cout<<"A"<<endl;}
4  };
5
6  class B : public A {
7      void print()
8      {cout<<"B"<<endl;}
9  };
10
11 A instance1;
12 instance1.print();
13 B instance2;
14 instance2.print();
15 A * pointer1 = &instance2;
16 pointer1->print();
```

Polymorphismus und dynamische Bindung

Ausgangspunkt: Abgeleitete Klasse die virtuelle Mitgliedsfunktion ihrer Basisklasse überschreibt.

Frage: Wann wird welche Version der Mitgliedsfunktion aufgerufen?

Faustregeln

- 1 Bei direktem Aufruf über eine Instanz wird immer die dem Typ entsprechende Mitgliedsfunktion aufgerufen.
- 2 Bei indirektem Aufruf über einen Pointer wird für nicht virtuelle Mitgliedsfunktionen immer die dem Typ des Pointers entsprechende Mitgliedsfunktion aufgerufen.
- 3 Bei indirektem Aufruf über einen Pointer wird für virtuelle Mitgliedsfunktionen immer die dem dereferenzierten Typ des Pointers entsprechende Mitgliedsfunktion aufgerufen.

Abstrakte Basisklasse

```
1  class A {  
2      virtual void print() = 0;  
3  };  
4  
5  class B : public A {  
6      void print()  
7      {cout<<"B"<<endl;}  
8  };  
9  
10 //A instance1; //Forbidden  
11 A * pointer1 = new B; //Allowed
```

Konstrukturen

```
1  class A {  
2      int a;  
3  public:  
4      A(int _a) : a(_a){}  
5  };  
6  
7  class B : public A {  
8      int b;  
9  public:  
10     B(int _a, int _b) : A(_a), b(_b) {}  
11 };
```

is a - has a

```
1  class point {
2  public:
3      double x;
4      double y;
5  };
6
7  class circle : private point {
8  private:
9      double radius;
10 };
```

is a - has a

```
1  class point {  
2  public:  
3      double x;  
4      double y;  
5  };  
6  
7  class circle {  
8  private:  
9      double radius;  
10     point p;  
11 };
```

is a - has a

```
1  class University {
2  private:
3      std::vector<Student>  students_;
4  };
5
6  class Student {
7  private:
8      Legi  legi_;
9  };
10
11 class Phys_Student : public Student {};
12
13 class Legi {
14     int  immatriculation_year_;
15 };
```


Prüfungsvorbereitung

Alte Prüfungen D-ITET

Alte Prüfungen D-PHYS