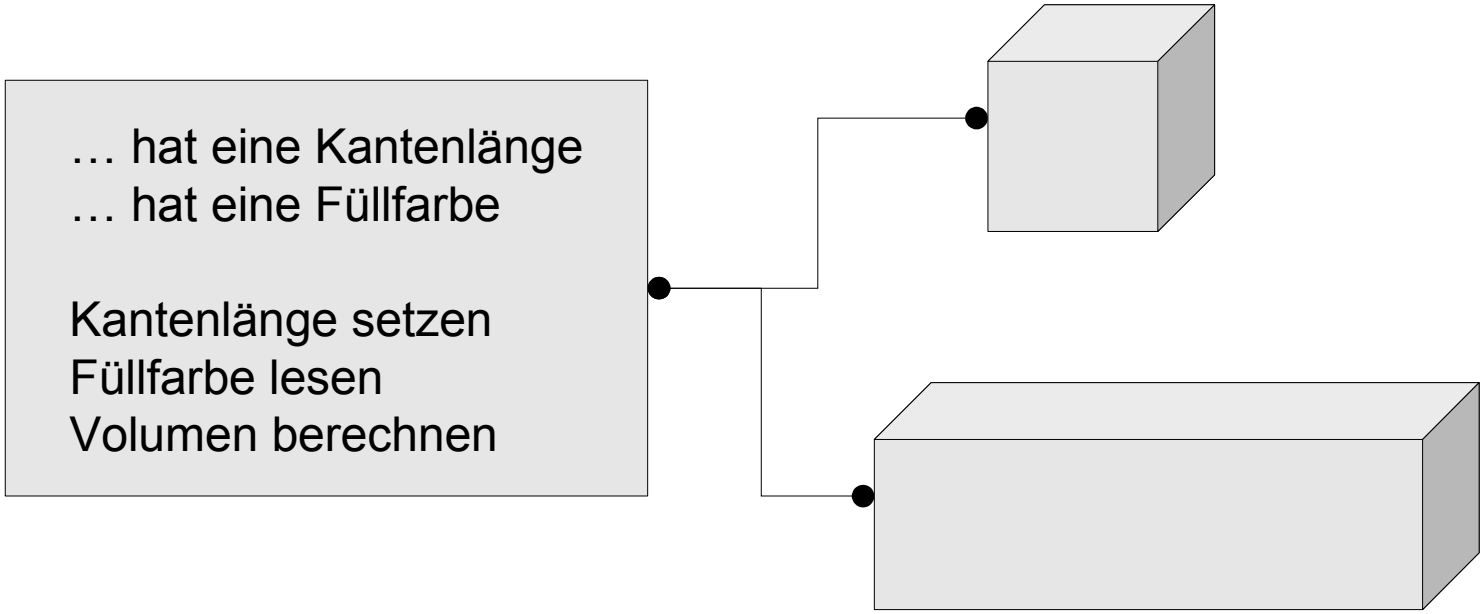
C++ - Objektorientierte Programmierung

Polymorphie



Polymorphie

- Zu einer Methode gibt es mehrere Implementierungen.
- Vielgestaltigkeit in Abhängigkeit von Basis- und Subklassen.
- Überschreibung von Methoden.
- Überladung von Methoden und Konstruktoren.

Überschreiben von Methoden aus der Basisklasse

- Eine Subklasse definiert eine Methode aus einer Basisklasse neu. Eine Methode aus der Basisklasse wird mit einer neuen Funktionalität versehen.
- Die Methoden haben den gleichen Namen, die gleiche Parameterliste und den gleichen Rückgabewert. Der Methodenkopf ist in der Basisklasse und Subklasse gleich.
- Die Methode in der Subklasse und die überschriebene Methode in der Basisklasse haben unterschiedliche Anweisungen im Methodenrumpf.

Methode in der Basisklasse

Beispiele/cppOOP_007_Ueberdecken...

```
string clsPerson::getInformationen()
{
    string strInfo;

    strInfo = this->getPersonName();
    strInfo.append("\n");
    strInfo.append(this->getAdresse());

    return strInfo;
}
```

Methode in der Subklasse

```
string clsMitarbeiter::getInformationen(){
    string strInfo;    stringstream streamInfo;

    streamInfo.str(getPersonName());
    streamInfo.seekp(0,std::ios::end);
    streamInfo << '\n' ;
    streamInfo.seekp(0,std::ios::end);
    streamInfo << this->getAdresse();
    streamInfo.seekp(0,std::ios::end);
    streamInfo << "\nGehalt: ";
    StreamInfo << std::fixed << std::setprecision(2) << gehalt;

    return streamInfo.str();
}
```

Beispiele/cppOOP_007_Ueberdecken...

Aufruf der Methode

```
class clsPerson person("Person Nachname", "vorname");
cout << "\nInformation" << person.getInformationen();

class clsMitarbeiter angestellte(mitarbeiter);
cout << "\nInformation: " << angestellte.getInformationen();
```

Beispiele/cppOOP_007_Ueberdecken...
..

- In Abhängigkeit des Objekttyps wird die entsprechende Methode gewählt.

Statische Bindung

- Standardbindung.
- Aufruf von Methoden in Abhängigkeit des Typs der Variablen oder Zeigers.
- Die Zuordnung zwischen der Objektvariablen und der Methode liegt zur Kompilierzeit fest.

Dynamische (späte) Bindung

- Anhand des Typs des Objekts wird entschieden, welche Methode aufgerufen werden soll.
- Anhand des Typs des Objekts, auf das der Zeiger oder die Referenz verweist, wird zur Laufzeit entschieden, welche Methode aufgerufen wird.
- Diese Bindungsart wird mit Hilfe des Schlüsselwortes `virtual` im Methodenkopf aktiviert.

Virtuelle Methoden

- Die Basisklasse stellt die Schnittstelle nach draußen zur Verfügung. Wenn die Implementierung in der Subklasse erfolgt, hat die Schnittstelle einen leeren Methodenrumpf.
- Die Subklasse kann die virtuelle Methode der Basisklasse überschreiben. Die Schnittstelle wird entsprechend der Funktionalität der Subklasse angepasst.

Definition in der Basisklasse

Beispiele/cppOOP_007_Virtual...

```
class clsGeometrie
{
  public:
      virtual int flaeche(){};
      virtual int umfang(){};

  protected:
      int hoehe;
      int breite;
};
```

Kopf einer virtuellen Methode

virtual	int	flaeche	(		)
virtual	Datentyp	name	(	Parameterliste	)

- Beginn mit dem Schlüsselwort virtual.
- Der Datentyp legt den Typ der Methode und des Rückgabewertes fest.
- Der Name der Methode ist eindeutig.
- In den runden Klammern folgt dem Methodennamen eine Parameterliste. Die Parameterliste kann leer sein.

Überschreibung in der Subklasse (Header-Datei)

Beispiele/cppOOP_007_Virtual...

```
class clsRechteck : public clsGeometrie
{
  public:
      int flaeche();
      int umfang();
};
```

Hinweis

- Methoden, die von der Subklasse überschrieben werden sollen, sollten in der Basisklasse immer mit dem Schlüsselwort virtual gekennzeichnet werden.
- Methoden, die nicht in der Basisklasse als virtuell gekennzeichnet sind, können sich bei einem Aufruf über eine Objektvariable anders als bei einem Aufruf mit Hilfe eines Zeigers verhalten.

C++11

Beispiele/cppOOP_007_Virtual...

```
class clsRechteck : public clsGeometrie
{
  public:
        int flaeche() override;
         int umfang() override;
};
```

Überprüfung des Compilers

- Der Compiler überprüft, ob in der Basisklasse eine entsprechende virtuelle Methode vorhanden ist.
- Die Methode ist mit Hilfe von override gekennzeichnet. Für, den in der Subklasse genutzten Kopf, muss es ein Pendant in der Basisklasse geben.
- Falls die zu überschreibende Methode in der Basisklasse nicht virtuell ist, wird ein Fehler gemeldet.

Überschreibung verhindern

```
virtual void zeichne() final{
    cout << '\n';

    for(int zeile = 1; zeile <= hoehe; zeile++)
    {
        for (int spalte = 1; spalte <= breite; spalte++)
        {
            cout << '*';
        }

        cout << '\n';
    }
}
```

Beispiele/cppOOP_007_Virtual...

Schlüsselwort final

- Die virtuelle Methode kann nicht überschrieben werden.
- Von der Basisklasse können keine Subklassen gebildet werden.

Virtuelle Destruktoren

```
virtual ~clsGeometrie()
{
}
```

Beispiele/cppOOP_007_Virtual...

- Basisklassen sollten immer einen virtuellen Destruktor haben.
- Mit Hilfe des virtuellen Destruktors wird zuerst der Destruktor der abgeleiteten Klasse und dann der Destruktor der Basisklasse aufgerufen.

Kopf eines virtuellen Destruktors

virtual	~clsGeometrie	(	)
virtual	~Klassenname	(	)

- Beginn mit dem Schlüsselwort virtual.
- Als Name für einen Destruktor wird immer der Name der Klasse, in der der Destruktor definiert ist, genutzt.
- Als Präfix vor dem Namen wird das Tilde-Zeichen gesetzt.
- Die runden Klammern zur Begrenzung der Parameterliste sind immer leer. Dem Destruktor werden keine Werte übergeben.

Aufruf des Destruktors

- Das Objekt wird zerstört. Beim Verlassen des Anweisungsblockes verliert die Objektvariable seine Gültigkeit.
- Durch das Schlüsselwort `delete` wird der entsprechende Destruktor aufgerufen.

Regeln

- Konstruktoren können nicht virtuell sein.
- Eine virtuelle Methode kann von der Subklasse überschrieben werden.
- Wenn eine Methode als virtuell deklariert ist, bleibt sie ab diesen Punkt für alle weiteren Spezialisierung auch virtuell.
- Die Parameterliste der virtuellen Methode in der Basisklasse und in der Subklasse müssen gleich sein.
- Eine virtuelle Methode, die in der Basisklasse als privat gekennzeichnet ist, kann in einer abgeleiteten Klasse überschrieben werden.

Pure virtuelle Methoden

- Die Basisklasse stellt den Prototypen der Methode bereit. Die Basisklasse implementiert die Methode aber nicht.
- Die Subklasse muss die Methode implementieren.

Definition in der Basisklasse

```
class clsGeometrie
{
  public:
      virtual clsGeometrie* create () const = 0;
      virtual clsGeometrie* clone ()  const = 0;

      virtual void getInformationen() = 0;

  protected:
      int hoehe;
      int breite;
};
```

Beispiele/cppOOP_007_Virtual...

Kopf einer puren virtuellen Methode

virtual	void	getInfo	(		)	=	0
virtual	Datentyp	name	(	Parameterliste	)	=	0

- Beginn mit dem Schlüsselwort virtual.
- Der Datentyp legt den Typ der Methode und des Rückgabewertes fest.
- Der Name der Methode ist eindeutig.
- In den runden Klammern folgt dem Methodennamen eine Parameterliste. Die Parameterliste kann leer sein.
- Der Methode wird der Wert 0 zugewiesen.

Abstrakte Basisklassen

- Definition mindestens einer puren virtuellen Methode.
- Schnittstellen-Definition für Subklassen.
- Eine Instanziierung von einer abstrakten Klasse ist nicht möglich.
- Eine, von einer abstrakten Klasse, abgeleiteten Klasse muss alle puren virtuellen Methoden definieren. Andernfalls kann von der Subklasse keine Instanz erzeugt werden.

Überschreibung in der Subklasse (Header-Datei)

```
clsRechteck* create () const  {
        return new clsRechteck;
}

 clsRechteck* clone () const {
        return new clsRechteck(*this);
 }

 void getInformationen() {
        cout << "\nBreite: " << breite;
        cout << "\nHoehe: " << hoehe;
 }
```

Beispiele/cppOOP_007_Virtual...

Überladen von Konstruktoren

- Konstruktoren haben den gleichen Namen.
- Entsprechend der Parameterliste bei der Initialisierung eines Objektes wird der Konstruktor aufgerufen.
- Alle Konstruktoren haben als Namen die Bezeichnung der Klasse, in der sie deklariert sind.
- Die Konstruktoren unterscheiden sich aber in der Parameterliste. Die Anzahl der Parameter und / oder die Datentypen der Parameter sind unterschiedlich.
- Konstruktoren geben kein Wert an den Aufrufer zurück.

… in einer Header-Datei

Beispiele/cppOOP_007_Ueberladen_...

```
class clsBankkonto {

public:
    clsBankkonto();
    clsBankkonto(double);
    clsBankkonto(double, int);
    clsBankkonto(const clsBankkonto& orig);

private:

};
```

Deklaration der Objektvariablen

Beispiele/cppOOP_007_Ueberladen_...

```
int main() {

    class clsBankkonto kontoA;

    class clsBankkonto kontoB(250);

    class clsBankkonto kontoC(30000, 1000);

    return 0;
}
```

Aufruf der Konstruktoren

class clsBankkonto kontoA;	clsBankkonto(){
	kontostand = 1;
	dispohoehe = 0;
	}

class clsBankkonto kontoB(250);	clsBankkonto(double guthaben)
	:kontostand(guthaben){
	dispohoehe = 0;
	}

class clsBankkonto kontoC
(30000, 1000);
	clsBankkonto::clsBankkonto
	(double guthaben, int dispo)
	:kontostand(guthaben)
	, dispohoehe(dispo){
	}

Überladen von Methoden

- Methoden haben den gleichen Namen. Verschiedene Methoden werden mit dem gleichen Namen aufgerufen.
- Die Methoden, die den gleichen Namen besitzen, unterscheiden sich aber in der Parameterliste. Die Anzahl der Parameter und / oder die Datentypen der Parameter sind unterschiedlich.
- Der Rückgabewert kann bei Methoden desselben Namens gleich sein, muss aber nicht.
- Der Compiler ruft entsprechend der Parameterliste die passende Methode auf.

… in einer Header-Datei

Beispiele/cppOOP_007_Ueberladen_...

```
class clsBankkonto {

public:
    void plusKontostand(double);
    void plusKontostand(clsBankkonto&, double);
    void minusKontostand(double);
    void minusKontostand(clsBankkonto&, double);

Private:

};
```

Überladen von Operatoren

- Nutzung von vorhandenen Operatoren für benutzerdefinierte Objekte.
- Die Funktionalität eines Operators wird für eine Klasse angepasst.

Regeln

- Es können keine neuen Operatoren definiert werden.
- Die Priorität und Assoziativität des Operators kann nicht verändert werden.
- Die Anzahl der Operanden kann nicht verändert werden.
- Mindestens einer der Operanden muss von einem benutzerdefinierten Typ (zum Beispiel `class`, `struct`) sein.
- Überladene Operatoren haben Zugriff auf private und geschützte Attribute.

Operatoren, die überladen werden können

- Alle Operatoren bis auf wenige Ausnahmen können überladen werden.
- Beispiele:

+	-					
*	/	%	+	-	++	--
<	<=	=>	>	==	!=	>>
&&	||	!	>	==	!=	>>
&	|	^	<<	>>	~	

Aufruf der Methode

```
class clsPerson person("Person Nachname", "vorname");
cout << "\nInformation" << person.getInformationen();

class clsMitarbeiter angestellte(mitarbeiter);
cout << "\nInformation: " << angestellte.getInformationen();
```

Beispiele/cppOOP_007_Ueberdecken...
..

- In Abhängigkeit des Objekttyps wird die entsprechende Methode gewählt.

Operatoren, die nicht überladen werden können

.	typeid	static_cast
.*	sizeof	const_cast
::		dynamic_cast
?:		reinterpret_cast

Operatoren, die nur in Klassen überladen werden

=	new	new[]	operator T() const; (Konvertierungsop.)
->	delete	delete()	
()			
[]			
->*			

Assoziativität von Operatoren

- Zum Beispiel „summe = a + b“.
- Die Assoziativität des Addieren-Operators legt fest, dass die Operanden von links nach rechts abgearbeitet werden. Der Wert a wird zu der Variablen b addiert.
- Die Assoziativität des Zuweisungsoperators legt fest, dass die Operanden von rechts nach links abgearbeitet werden. Der Ausdruck rechts vom Zuweisungsoperator wird im ersten Schritt ausgewertet und anschließend das Ergebnis der Variablen links vom Operator zugewiesen.

Priorität von Operatoren

- Zum Beispiel „summe = a + b“.
- Der Addieren-Operator hat eine höhere Priorität als der Zuweisungsoperator. Im ersten Schritt werden a und b addiert. Im zweiten Schritt wird der berechnete Wert der Variablen summe zugewiesen.

Operator-Funktionen

- Beispiel: operator+ (a, b)
- Die Funktionsname beginnt mit der Bezeichnung operator. Der Bezeichnung folgt der Operator.
- Der Funktion können in runden Klammern Parameter übergeben werden. In diesem Beispiel werden die zu addierenden Parameter übergeben.
- Die Operator-Funktion gibt einen Wert zurück.

Überladung des Zuweisungsoperators

Beispiele/cppOOP_007_UeberladenOperatoren...

```
clsBankkonto& operator= (clsBankkonto&);
```

```
clsBankkonto& clsBankkonto::operator= (clsBankkonto& quelle){
    this->kontostand = quelle.kontostand;
    this->dispohoehe = quelle.dispohoehe;

    return *this;
}
```

Prototypen

```
clsBankkonto& operator= (clsBankkonto&);
```

```
referenz operator= (referenz);
```

- Der Standard-Zuweisungsoperator weist einer Objektvariablen die, in der anderen Objektvariablen gespeicherten Adresse zu. Beide Objektvariablen zeigen auf das gleiche Objekt.
- Wenn ein Kopierkonstruktor, ein Destruktor oder ein überladender Zuweisungsoperator implementiert ist, werden auch alle anderen Methoden benötigt.

Parameter

ClsBankkonto& operator= (clsBankkonto&);
klasse operator= (referenz);

- Objekte werden als Referenz übergeben.

Rückgabewert

```
clsBankkonto& clsBankkonto::operator= (clsBankkonto& quelle){
    this->kontostand = quelle.kontostand;
    this->dispohoehe = quelle.dispohoehe;

    return *this;
}
```

- Die Funktion des Zuweisungsoperators liefert eine Referenz auf das aktuelle Objekt zurück.
- Mit Hilfe der Anweisung `return *this` wird eine Referenz zurückgeliefert.

this

- Zeiger auf das aktuelle Objekt.
- Das Schlüsselwort this ist ein Platzhalter für das Objekt, welches die Methode oder den Konstruktor aufgerufen hat.
- Mit Hilfe von this->attribut wird die Instanzvariable des aktuellen Objekts verändert.
- *this dereferenziert die Adresse des aktuellen Objekts.

Nutzung

- Rückgabe des aktuellen Objekts durch *this.
- Beim Kopieren von Attribut-Werten von einem Objekt in das aktuellen Objekt wird this->attribut benötigt.
- Mit Hilfe der Anweisung this != adresse kann ein Verweis auf sich selbst ausgeschlossen werden.

Aufruf der überladenen Operatormethode

```
kontoB = kontoA;
```

```
kontoA = (kontoA - 20);
```

- Die überladene Methode wird durch Anwendung des Operators aufgerufen.

Prototyp des Operators „Subtraktion“

```
clsBankkonto& operator- (double);
```

Beispiele/cppOOP_007_UeberladenOperatoren...

- operator- (a, b). result = a – b.
- Der erste Operator ist automatisch ein Objekt der eigenen Klasse. Das Objekt wird nicht als Parameter benötigt.

Implementierung in der Quelldatei

```
clsBankkonto&  clsBankkonto::operator- (double betrag){
    double neuKontostand;

    neuKontostand = this->kontostand - betrag;

    if (neuKontostand < 0){

    }
    else {
            this->kontostand = neuKontostand;
    }

    return *this;
}
```

Beispiele/cppOOP_007_UeberladenOperatoren...

Rückgabewert

```
clsBankkonto&  clsBankkonto::operator- (double betrag){
    double neuKontostand;

    neuKontostand = this->kontostand - betrag;

    return *this;
}
```

- Die Funktion des Operators liefert einen Verweis auf das aktuelle Objekt zurück.
- Mit Hilfe der Anweisung `return *this` wird ein Verweis auf das aktuelle Objekt zurückgeliefert.

Aufruf der überladenen Operator-Methode

```
kontoA = (kontoA - 20);
```

- Die überladene Methode wird durch Anwendung des Operators aufgerufen.
- Der linke Operator ist ein Objekt von der eigenen Klasse des überladenen Operators.

Vergleich von Objekten

```
kontoA == kontoB
```

Beispiele/cppOOP_007_UeberladenOperatoren...

- Der Operator „ist gleich“ muss überladen werden, um die Instanzvariablen auf Gleichheit zu überprüfen.
- Der Standardoperator vergleicht die, in den Objektvariablen gespeicherten Adressen, aber nicht das Objekt.

Prototyp des Operators „ist gleich“

```
bool operator==(clsBankkonto&);
```

Beispiele/cppOOP_007_UeberladenOperatoren...

- Vergleichsoperatoren liefern immer einen boolschen Wert zurück.

Implementierung in der Quelldatei

Beispiele/cppOOP_007_UeberladenOperatoren...

```
bool clsBankkonto::operator==(clsBankkonto& konto)
{
   bool isGleich = false;

   if (konto.kontostand == this->kontostand)
   {
      if(konto.dispohoehe == this->dispohoehe)
      {
         isGleich = true;
      }
   }

   return isGleich;
}
```