Polymorphismus und seine Bedeutung in der Anwendung

Probevorlesung an der
Fachhochschule Westküste

Ulrich Hoffmann (uho@xlerb.de)

Polymorphismus ist die objekt-orientierte Art Software–Schnittstellen zu realisieren

Überblick

- Eine einfache technische Beispielanwendung, die Polymorphismus benutzt
- Objekte, Klassen und Methoden
- Vererbung
- Polymorphe Zuweisungen
- Überschreiben von Methoden
- Methoden–Auswahl, statisches und dynamisches Binden
- Bedeutung des polymorphen Methodenaufrufs
- Erweiterungen der Beispielanwendung

Polymorphismus: eine einfache Beispielanwendung

- Polymorphismus bedeutet allgemein *Vielgestaltigkeit*
- engere Bedeutung: spezielle Form der Methodenauswahl in objekt-orientierten Sytemen
- Polymorphismus ist die objekt-orientierte Art Software–Schnittstellen zu realisieren

Eine einfache Beispielanwendung:

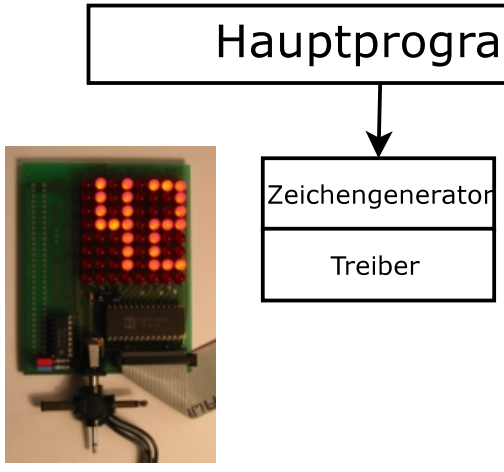
- Ausgabe in eingebetteten Systemen
- Darstellung natürlicher Zahlen von 0 bis 99
- Programmiert in C++ unter Einsatz von Polymorphismus
- Beispiel hier:
  - 8x8 LED-Feld zur Ausgabe von Daten
  - angeschlossen über Centronics-Schnittstelle

Polymorphismus: eine einfache Beispielanwendung

Grobe Programmstruktur:

1. Hauptprogramm: Ablaufsteuerung, steuert Zahlenausgabe
2. Ausgabe auf dem LED-Feld
   - Darstellung der Ziffern und Zahlen (Zeichengenerator)
   - Ansteuerung der einzelnen Leuchtdioden (Treiber)

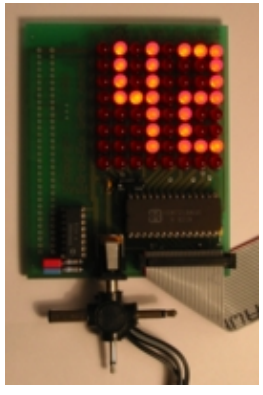
LED-Feld besitzt eine (einfache) objekt-orientierte Software–Schnittstelle, die vom Hauptprogramm benutzt wird.

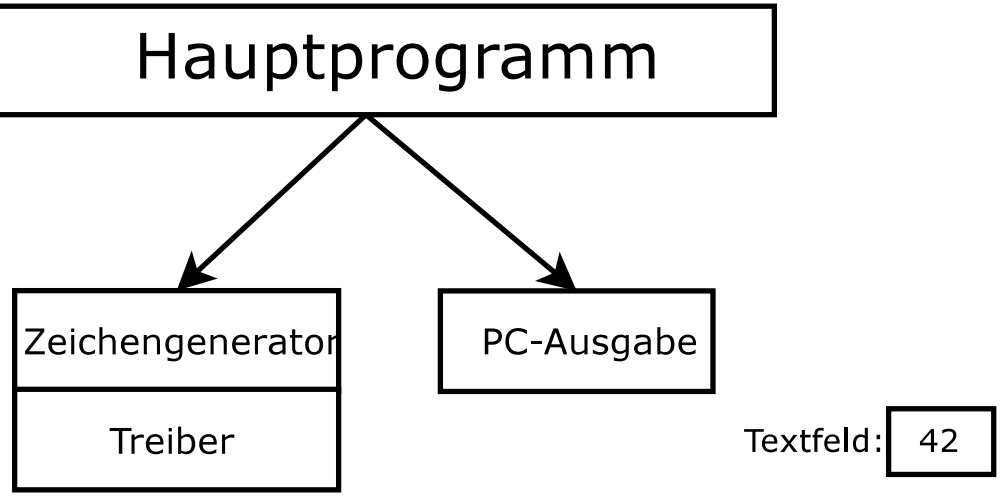


Polymorphismus heißt Vielgestaltigkeit

- Durch klare Schnittstelle: einfacher Austausch der Zahlenausgabe möglich
- Ausgabe des Zahlenwertes auf dem PC
- Gleichzeitiger Betrieb von LED-Feld und PC-Ausgabe

LED-Feld **und PC-Ausgabe** besitzen **die selbe** (einfache) objekt-orientierte Softwareschnittstelle, die vom Hauptprogramm **polymorph** benutzt wird.





Wo im Studienplan würde diese Vorlesung stattfinden?

1. Semester: Vorlesung "Grundlagen der Informatik 1" + Praktikum

u. a. Einführung Programmieren in C++ (Schwerpunkt prozedurale Aspekte) das C an C++ Zahlentypen, Variablen, Ausdrücke, Zuweisungen, Fallunterscheidungen, Schleifen, Funktionen, Programmlayout und Programmstil, höhere Datentypen: Arrays & Structs, Pointer, Bitstruktur der Daten, Dynamische Datenstrukturen, außerdem: Elemente der Rechnerhardware

2. Semester: Vorlesung "Grundlagen der Informatik 2" + Praktikum

u. a. Programmieren in C++ (Schwerpunkt objekt-orientierte Aspekte)

- Klassen und Objekte
- Methoden (Member Functions) und Sichtbarkeiten (private, protected, public)
- Vererbung
- Methoden–Dispatch (statisches dynamische Binden, virtual, static)
- Polymorphismus
- Klassendesign, Datenstrukturen und Algorithmen
- Windows-Programmierung oder Programmierung eingebetteter Systeme

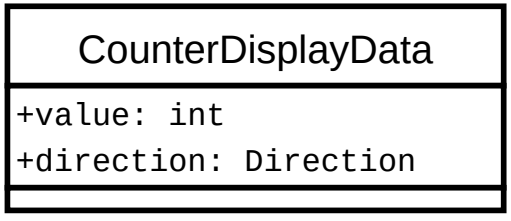
außerdem: PC-Hardware und Datennetze

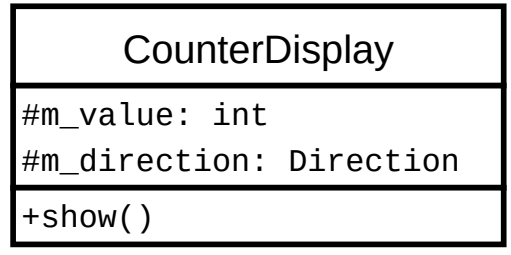
Und dann noch: abstrakte Funktionen (pure virtual), Upcasts, Mehrfachvererbung, Templates, Standard Template Library, Funktionsobjekte, Exceptions, ...

Objekte, Klassen und Methoden in einer Folie

```
struct CounterDisplayData {
   int value;
   Direction direction;
}
```

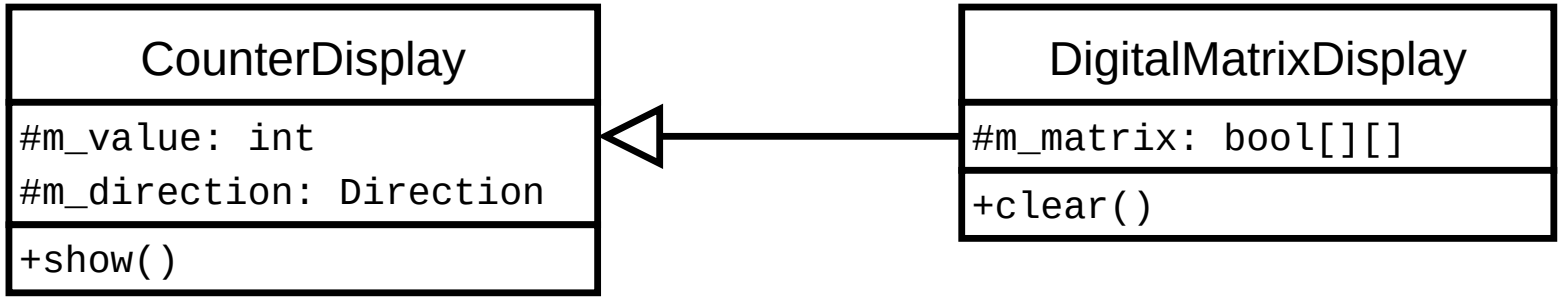
```
class CounterDisplay {
protected:
   int m_value;
   Direction m_direction;
public:
   void show();
}
```





- Klassen in C++ sind Erweiterungen von Structs.
- Structs haben Datenfelder (*Member Variablen*, *Instanzvariablen*, *Attribute*).
- neu: Klassen haben auch zugehörige Funktionen (*Member Funktionen*, *Methoden*).
- Kapselung (encapsulation), Geheimnisprinzip (information hiding)
- Zugriff auf Felder (Member):
  `d.m_value` und `d.show()` wenn p Zeiger auf Objekt: `p->m_value` bzw. `p->show()`

Vererbung in einer Folie



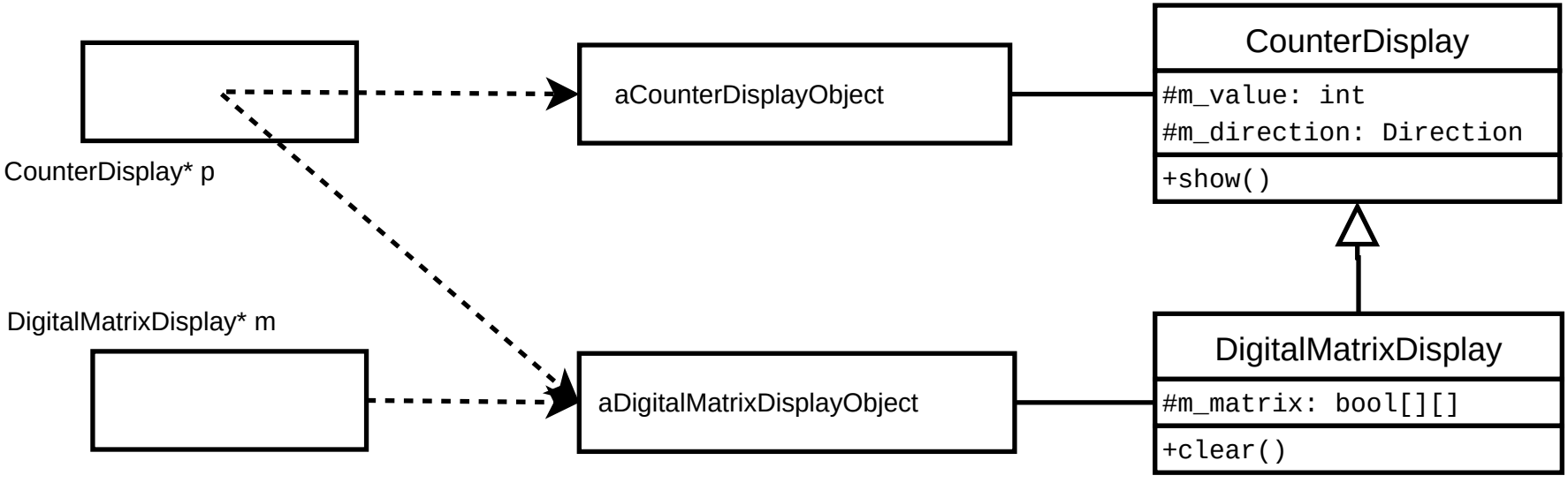

```
class DigitalMatrixDisplay : public CounterDisplay { ... };
```

- Klassen werden in Beziehung gesetzt.
- Member der *Basisklasse* (Super/Oberklasse) sind auch in den abgeleiteten Klassen verfügbar.
  - Vorteil: Gemeinsamkeiten werden nur einmal programmiert.
    Wiederverwendung von Implementierungen
  - Vorteil: Objekte der abgeleiteten Klasse können wie Objekte der Basisklasse behandelt werden.
    Basisklasse bildet Schnittstelle für alle ihre Unterklassen.
    Realisierung von Generalisierungs–/Spezialisierungs–Beziehungen
    wichtig im kaufmännischen und im technischen Bereich (z.B. Gerätetreiber)

Polymorphe Zuweisungen

- Zuweisungen bisher: var = expr;
- var und expr müssen den **gleichen** Datentyp besitzen
  - Ausnahme Zahlentypen, Dualität zwischen Arrays und Pointern
- Zuweisungen jetzt: var = expr;
- Wenn var vom Typ Zeiger auf (Objekt der) Basisklasse ist, darf expr auch vom Typ Zeiger auf (Objekt einer) abgeleiteten Klasse sein.
- Folge: Variablen haben jetzt zwei Typen
  1. statischer Typ: der deklarierte Typ
  2. dynamischer Typ: der Typ des Wertes der in der Variablen gespeichert ist

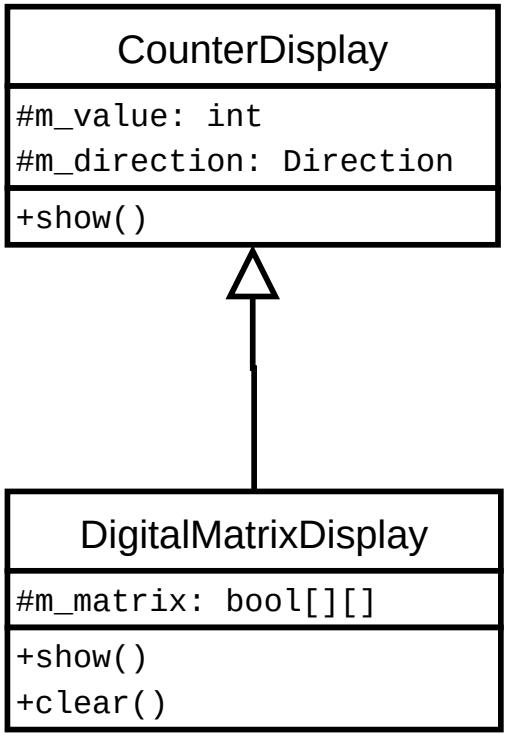
Polymorphe Zuweisungen




```
CounterDisplay* p;
DigitalMatrixDisplay* m;

p=new CounterDisplay;           // ok
p=new DigitalMatrixDisplay;     // ok
m=new CounterDisplay;           // Fehler
m=new DigitalMatrixDisplay;     // ok
```

Überschreiben von Methoden (Overriding)

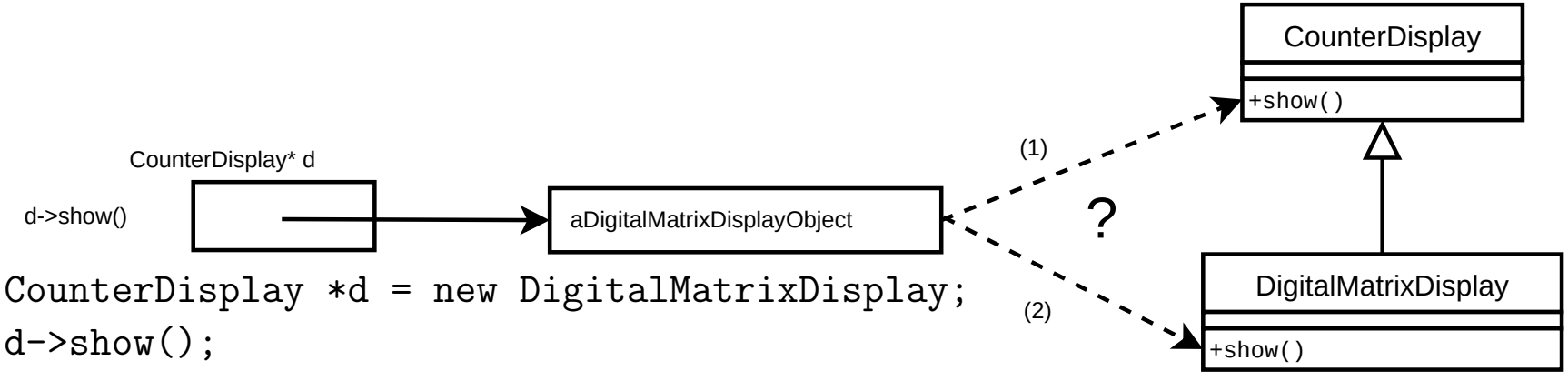



```
class CounterDisplay {
...
void show();
};
...
void CounterDisplay::show() { ... }

class DigitalMatrixDisplay : public CounterDisplay {
...
void show();
};
...
void DigitalMatrixDisplay::show() { ... }
```

- In abgeleiteten Klassen können Methoden der Basisklasse neu definiert (*überschrieben*) werden.
- Parametertypen und der Ergebnistyp müssen übereinstimmen.
- Achtung: nicht verwechseln mit Überladen von Funktionen (unterschiedliche Parametertypen).

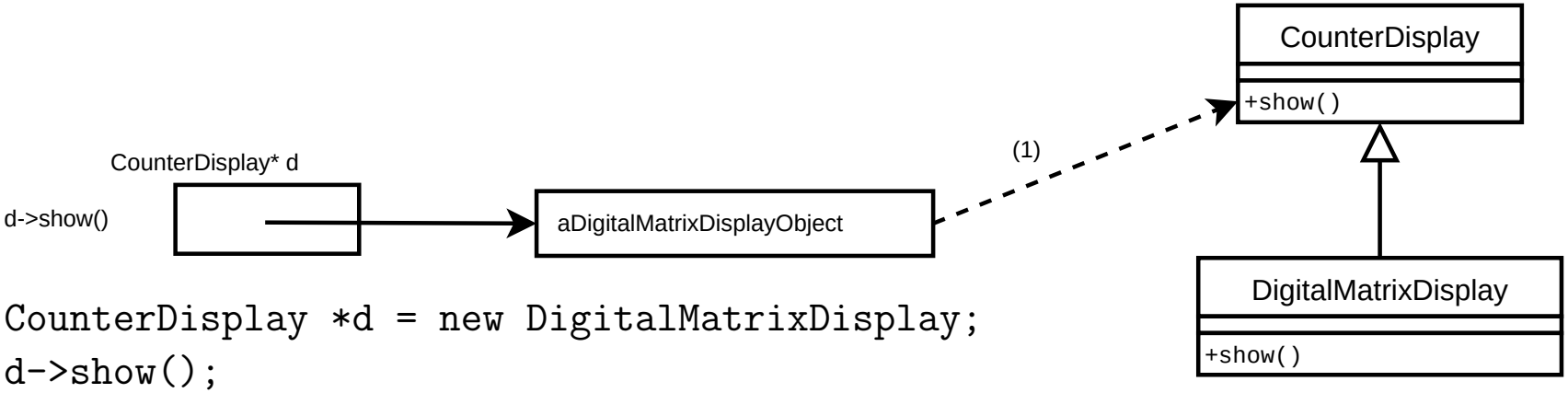
Methoden–Auswahl (Dispatch)




```
CounterDisplay *d = new DigitalMatrixDisplay;
d->show();
```

- Frage: Welcher Funktionsrumpf der Methode `show` wird ausgeführt? (1) oder (2)
- Antwort: abhängig davon, ob `show` eine virtuelle Funktion ist oder nicht (Schlüsselwort `virtual`).

Methoden–Auswahl (statisches Binden)



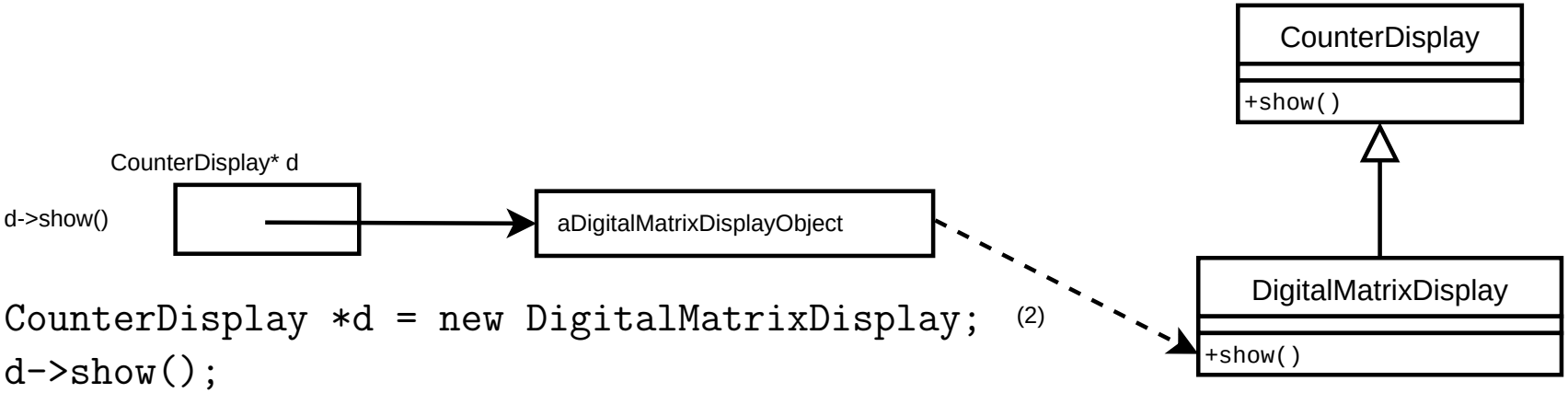

```
CounterDisplay *d = new DigitalMatrixDisplay;
d->show();
```

(1) wird ausgeführt wenn `show` **keine** virtuelle Funktion ist (*statisches Binden*).

```
class CounterDisplay { ...
   void show();
... }
```

Wenn Methode in der Basisklasse nicht definiert: Übersetzungszeitfehler.

Methoden–Auswahl (dynamisches Binden)



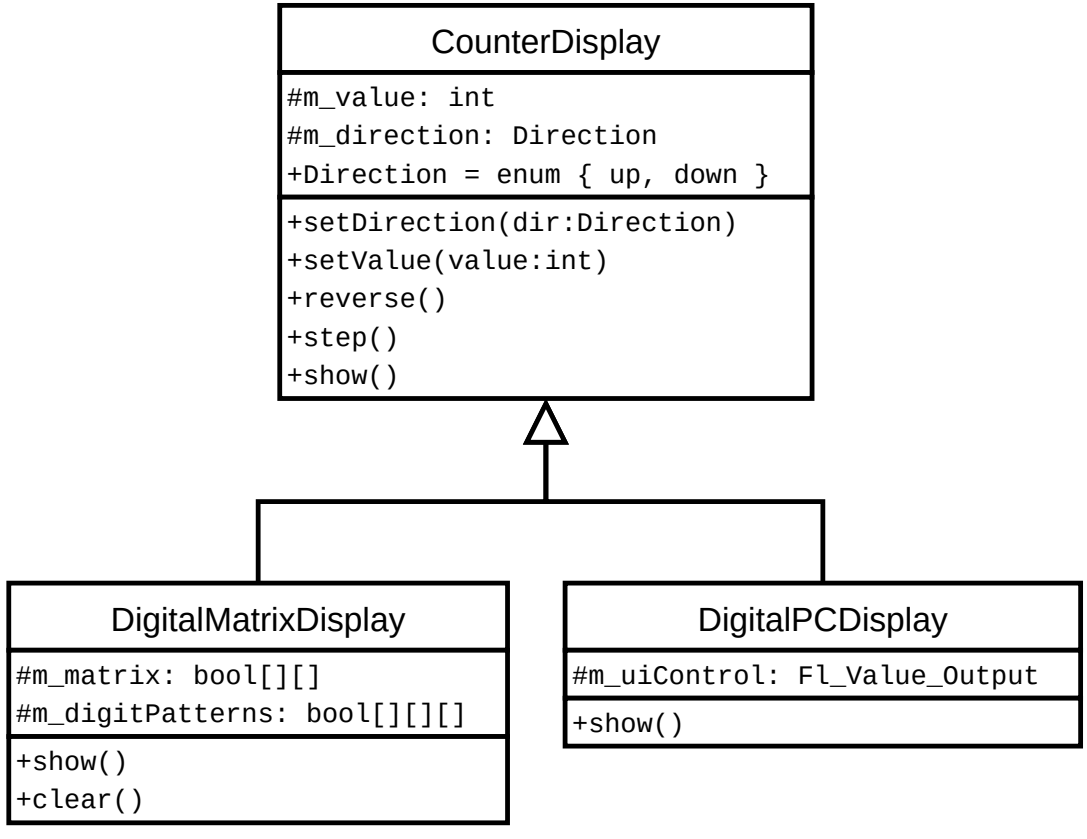

```
CounterDisplay *d = new DigitalMatrixDisplay;  (2)
d->show();
```

(2) wird ausgeführt, wenn `show` **eine** virtuelle Funktion ist (*dynamisches Binden*).

```
class CounterDisplay {  ...
   virtual void show();
... }
```

Wenn Methode in Basisklasse nicht definiert: Übersetzungszeitfehler.
Wenn Methode in abgeleiteter Klasse nicht definiert: Suche im Vererbungsgraphen von der abgeleiteten Klasse an aufwärts bis zur Basisklasse, erster gefundener Methodenrumpf wird ausgeführt.

PC-Ausgabe: zweite Implementierung von CounterDisplay



Bedeutung des polymorphen Methodenaufrufs

Beim Übersetzen des Aufrufs einer polymorphen Funktion ist i. A. nicht festgelegt, welche konkrete Implementierung ausgeführt werden wird. Dies wird erst zur Laufzeit bestimmt.

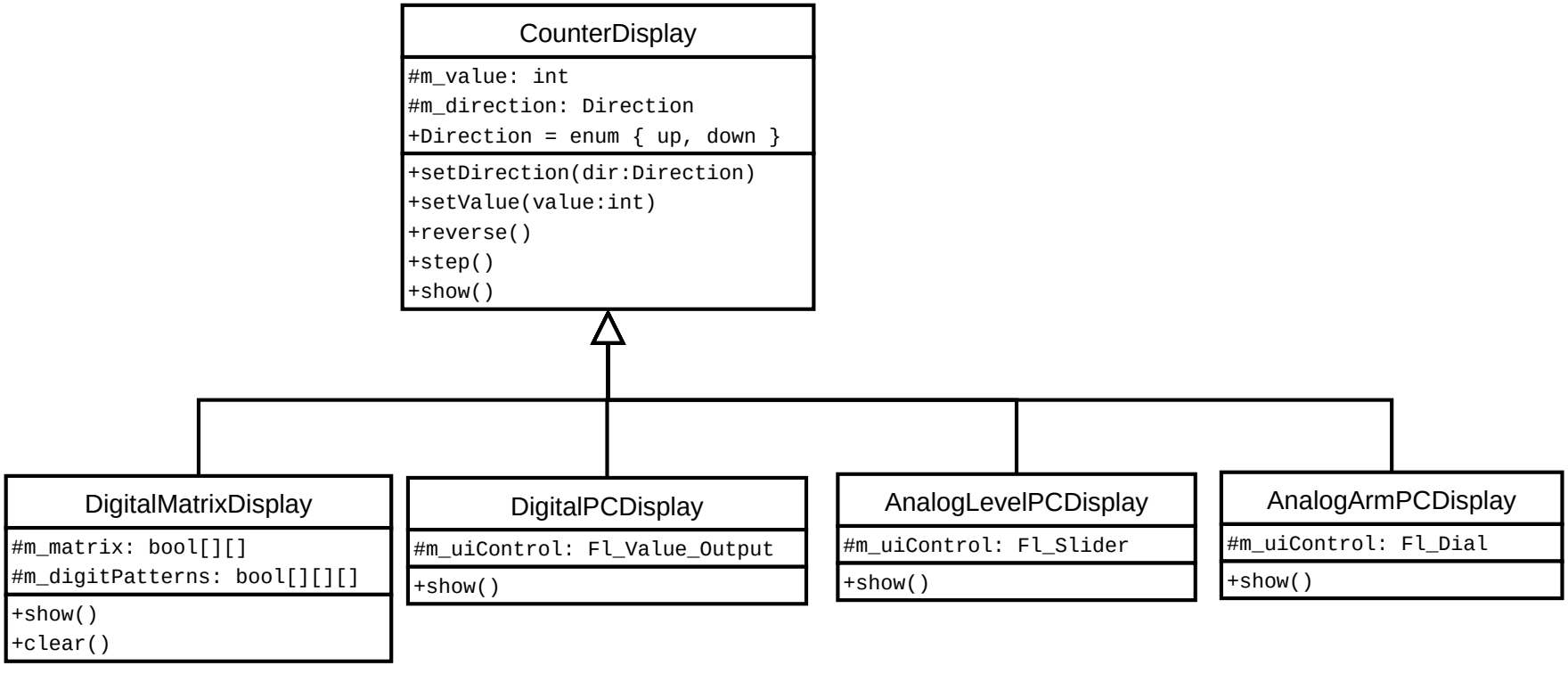
Konsequenzen für den Programmentwurf

- Erweiterbarkeit
  - Programm kann später durch weiteres Vererben von der Basisklasse geändert werden, ohne dass die Aufrufstelle geändert werden oder neu übersetzt werden muß.
  - Erweiterbarkeit von Binärcode durch Hinzufügen weiteren Codes (DLLs). Plug-Ins.
  - Kann auch nicht objekt-orientiert durch C–Funktionspointer erreicht werden.
  - Für kommerzielle Systeme von Bedeutung; Beispiel: Adobe Produkt-Familie
  - In eingebetten Systemen von untergeordneter Rolle, da Binärcode in der Regel vollständig neuübersetzt wird.

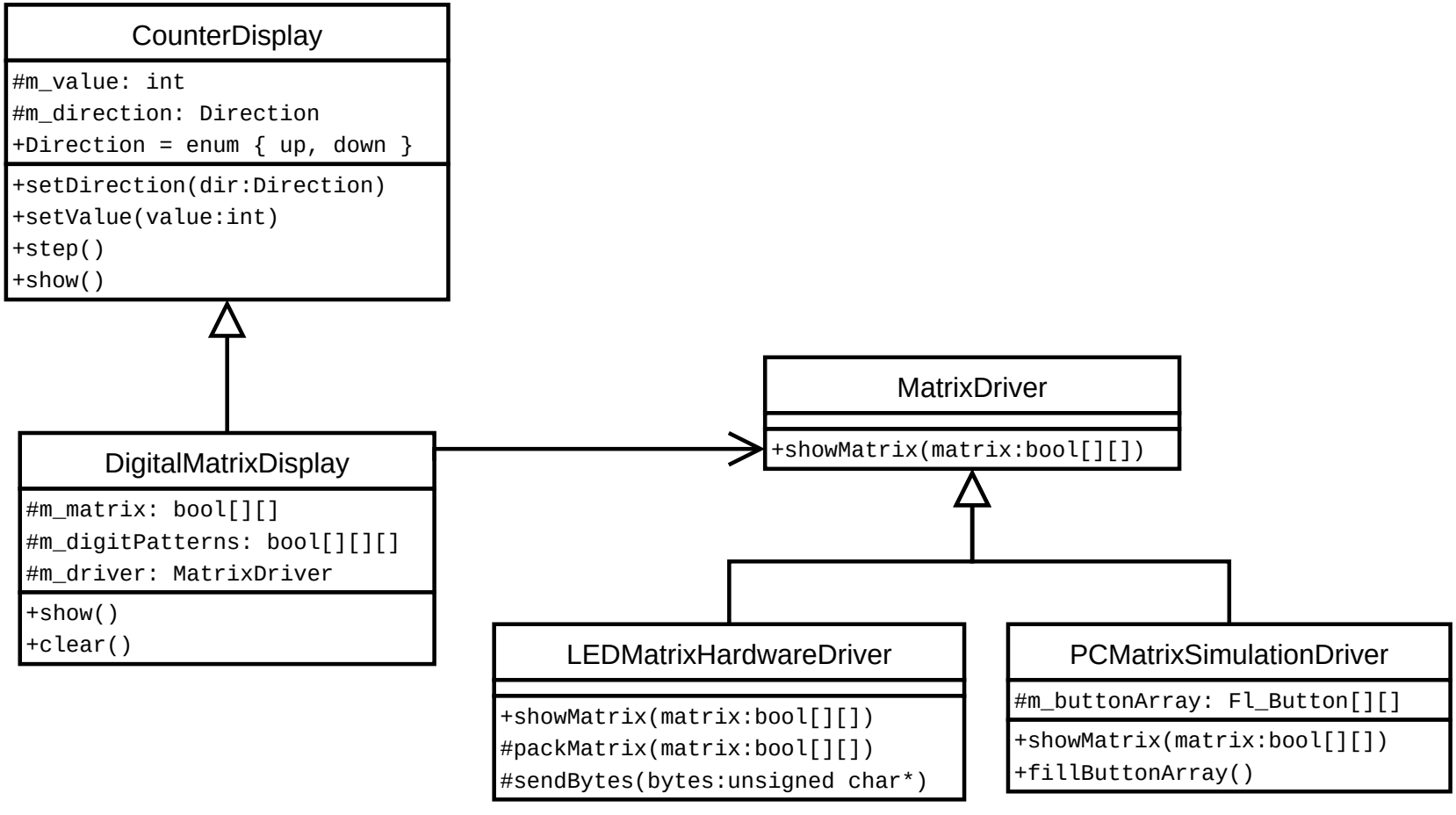
Bedeutung des polymorphen Methodenaufrufs

- Schnittstellen
  - Methode in der Basisklasse bildet Schnittstelle für den Aufrufer
  - legt Parameter, Resultat und Verhalten fest
  - Aufrufer kann mit dieser Schnittstelle arbeiten, ohne die Details der Realisierung zu kennen.
  - Für eingebettet Systeme wichtig: Treiber-Konzept - Eine Schnittstelle viele mögliche Implementierungen.
  - Dies kann und wird in C auch über Funktionspointer realisiert. In C++ wird dies direkt unterstützt.
  - Polymorphismus ist die objekt–orientierte Art Software–Schnittstellen zu definieren.

Weitere polymorphe Implementierungen



Ein weiterer Ansatzpunkt für Polymorphismus



Zusammenfassung und Ausblick

Zusammenfassung:

- Schnelleinführung in objekt-orientierte Begriffe
- Polymorphismus: Polymorphe Zuweisungen
- Polymorphismus: Methoden–Auswahl, statisches und dynamisches Binden
- zwei Stellen des Einsatzes von Polymorphismus in einem einfachen technischen Beispielsystem

Ausblick:

- Abstrakte Klassen und Methoden, rein virtuelle Funktionen (pure virtual functions)
- Realisierung einer Level–Anzeige auf dem LED-Feld
- Ausgabe über analoges Gerät
- Software–Probleme durch Polymorphismus

  Polymorphismus ist die objekt-orientierte Art Software–Schnittstellen zu realisieren

- Vortragsfolien erhältlich unter www.xlerb.de
- Fragen