# Praxis der Programmierung

## Kopieren von Objekten, abstrakte Klassen

Institut für Informatik und Computational Science
Universität Potsdam

Henning Bordihn

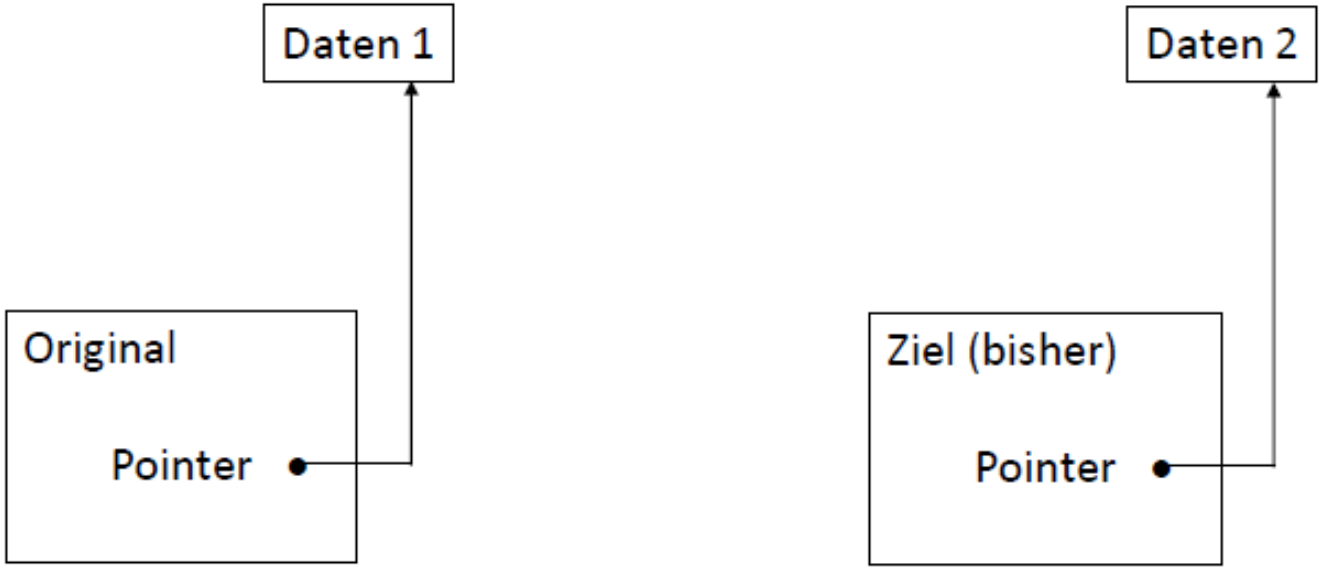
# Kopieren von Objeken

# Wann werden Objekte kopiert?

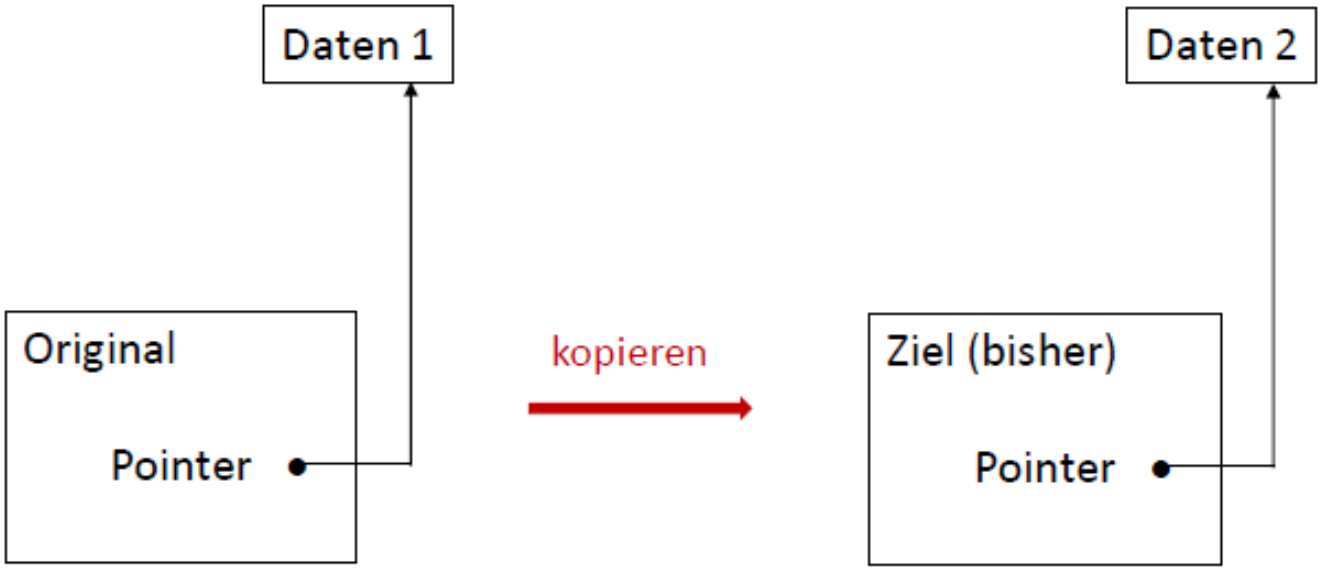
- Zuweisung `objekt1 = objekt2;`
- Objekte als Parameter von Methoden/Funktionen
  (call by value!!!)
- Objekte als Rückgabewerte von Funktionen

**Dabei werden Objekte bitweise kopiert.**
**⇝flache Kopie**

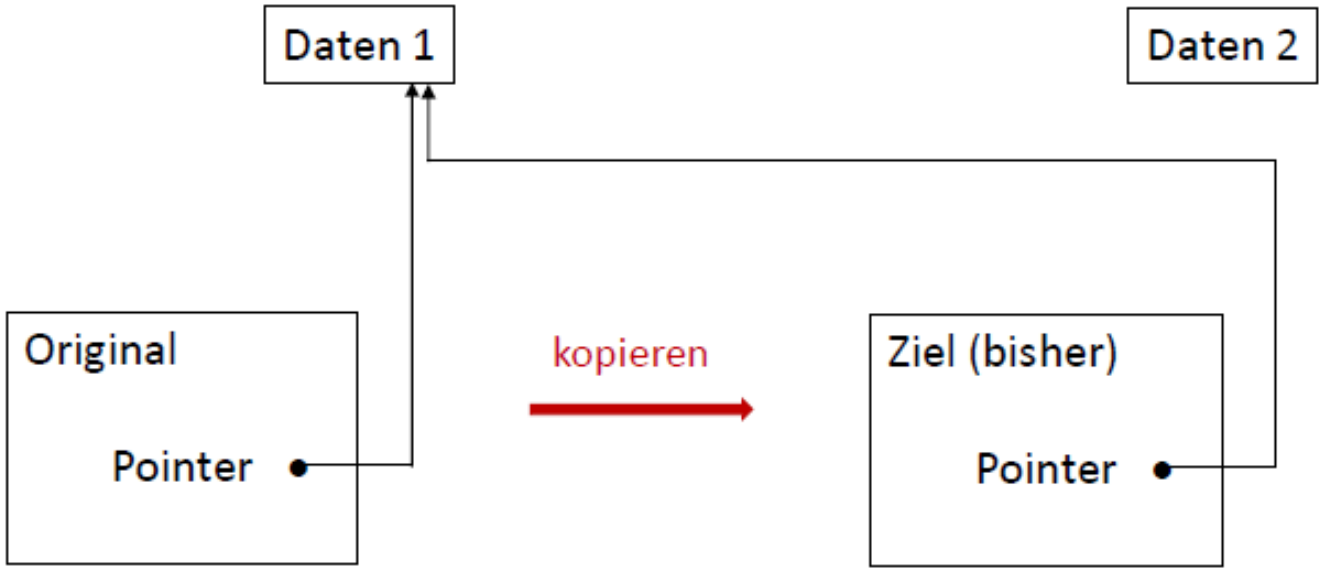
# Flache Kopien



# Flache Kopien



# Flache Kopien



# Tiefe Kopie mittels Kopierkonstruktor

## neuen Pointer anlegen und Wert an neue Adresse kopieren

```
class Cls {
    int * ptr;   // referenzierte Daten
    int n;       // sonstige Daten
public:
    // Kopierkonstruktor
    Cls(Cls * orig) {
        ptr = new int;
        *ptr = *orig->ptr;
        n = orig->n;
    }
    void setData(int value) {
        *ptr = value;
    }
};
```

# Tiefe Kopie mittels Kopierkonstruktor

## neuen Pointer anlegen und Wert an neue Adresse kopieren

```
class Cls {
    int * ptr;   // referenzierte Daten
    int n;       // sonstige Daten
public:
    // Kopierkonstruktor
    Cls(Cls * original) {
        ptr = new int;
        *ptr = *orig->ptr;
        n = orig->n;
    }
    void setData(int value) {
        *ptr = value;
    }
};
```

```
int main() {
   Cls obj;
   obj.setData(16);
   Cls flat = obj;
   Cls deep(&obj);
   obj.setData(66);

   // *(obj.ptr) = 66
   // *(flat.ptr) = 66
   // *(deep.ptr) = 16
```

# Referenzen und Referenzparameter

- Referenz: Variable, die wie ein *Aliasname* auf ein Speicherobjekt verweist
- keine Adresse (Pointer), sondern ein (zweiter) Name, mit dem auf die Speicherstelle zugegriffen wird
- Referenzparameter: Übergabe der Referenz
  $\rightsquigarrow$ Manipulation am Speicherobjekt möglich
- Definition: *Datentyp* & *Bezeichner*
- Beispiel: `int & reference`

```
void inc(int & n) {
    n++;
}
```

```
int main() {
     int number = 22;
     inc(number);      // number = 23
 }
```

# Referenz- *versus* Pointerparameter

- Pointer können verändert werden, also später auf andere Speicherobjekte zeigen
- Referenzen können nach der Parameterübergabe nicht mehr auf andere Speicherstellen "umgebogen" werden
- aktueller Parameter bei Referenzen: (dereferenzierte) Variable (kein Pointer!)

$\rightsquigarrow$ Aufrufer sieht nicht, ob die Variable als Wert oder Referenz übergeben wird
$\rightsquigarrow$ *auf mögliche Seiteneffekte achten!!!*

# Kopierkonstruktor mit Referenz auf das Original

```
class Cls {
    int * ptr;   // referenzierte Daten
    int n;       // sonstige Daten
public:
    // Kopierkonstruktor
    Cls(Cls & original) {
        ptr = new int;
        *ptr = *orig.ptr;
        n = orig.n;
    }
    void setData(int value) {
        *ptr = value;
    }
};
```

```
int main() {
   Cls obj;
   obj.setData(16);
   Cls flat = obj;
   Cls deep(obj);
   obj.setData(66);

   // *(obj.ptr) = 66
   // *(flat.ptr) = 66
   // *(deep.ptr) = 16
```

# Aufgabe 1

Ergänzen Sie Ihre Klasse `HighScore` um einen Kopierkonstruktor.
Verwenden Sie einen Referenzparameter.

Testen Sie ihn mit Ihrer Beispielanwendung `useHighScore.cpp`.

# Lösung Aufgabe 1

```
HighScore(HighScore & orig) {
   score = orig.score;
   date = new Date();
   *date = *orig.date;
}

int main() {
   HighScore hsc;
   // ... Daten setzen
   HighScore cpy(hsc);
   // ... hsc veaendern
   // ... Daten von hsc und cpy ausgeben
}
```

# Abstrakte Klassen

# Abstrakte Klassen als abstrakte Oberbegriffe

- Quadrate, Kreise, Rechtecke, Dreiecke, ...
  ... sind *ebene Figuren*

- Idee: gemeinsame Oberklasse `Figure`

- aber: keine Exemplare von `Figure` (*abstrakt!*)

# Design von `Figure`

- gemeinsame Datenelemente (`Point`)
- gemeinsame Methoden (Getter und ggf. Setter für gemeinsame Datenelemente, `moveTo()`, `moveRel()`)
- gemeinsame Methoden**signaturen** (area(), perimeter(), ...)
  $\rightsquigarrow$ Implementierung??? ... keine!!!
- `virtual double area() = 0;`
  - abstrakte Methoden: virtuelle Methoden, denen 0 zugewiesen wird
  - Implementierung erfolgt in Unterklassen
  - abstrakte Klasse: Klasse mit mindestens einer abstrakten Methode
    $\rightsquigarrow$ keine Exemplare erzeugbar

# Aufgabe 2

1. Erstellen Sie die abstrakte Klasse `Figure`.
   - Benutzen Sie `Point` (als Datenelement).
   - Definieren Sie `area()` und `perimeter()` als abstrakte Methoden.
   - Trennen Sie die Klassendefinition und die Implementierung des Verhaltens.

2. Modifizieren Sie `Square` so, dass sie implementierte Unterklasse von Figure ist.

3. Schreiben Sie eine kleine Anwendung, mit der Sie diese Klassen testen.