

# Initialisierung und Finalisierung von Objekten

Vorkurs C/C++, Olaf Bergmann

- Objekt anlegen und benutzen  
`Point pnt;`  
`pnt.draw();`
- Wie wird ein Objekt der Klasse `Point` initialisiert?

```
class Point {  
protected:  
    double _x, _y;  
public:  
    void draw();  
};
```

```
class Point {  
protected:  
    double _x, _y;  
public:  
    Point(double x = 0, double y = 0);  
    void draw();  
};
```

```
Point::Point(double x, double y) : _x(x), _y(y) {}
```

```
Point null, pnt(100.5, 53);  
  
null.draw();  
pnt.draw();
```

- Kein Konstruktor angegeben  
→ automatisch erzeugter Default-Konstruktor
- sonst: explizite Definition erforderlich

```
class Point {  
protected:  
    double _x, _y;  
public:  
    Point(double x, double y) : _x(x), _y(y) {}  
    void draw();  
};  
  
Point pnt;                // Error  
Point null(0,0);         // Ok
```

- Erzeugt 1:1-Kopie eines Objekts (→ shallow copy)

```
class Point {  
protected:  
    double _x, _y;  
public:  
    Point(double x = 0, double y = 0);  
    void draw();  
};
```

```
Point null;  
Point p(null);
```

```
p._x = null._x;  
p._y = null._y;
```

**b.items = a.items**  
→ beide Member zeigen nun  
auf dieselbe Speicheradresse!

```
class MyArray {  
    Object *items;  
    size_t count;  
public:  
    void free_all();  
};
```

```
MyArray a;  
/* a befüllen ... */  
MyArray b(a);  
a.free_all();
```



- Copy-Konstruktor anpassen oder verbieten

```
class MyArray {
    Object *items;
    size_t cnt;
public:
    MyArray() : items(nullptr),
               cnt(0) {}
    MyArray(const MyArray &);
    void free_all();
};

MyArray::MyArray(const MyArray &a) {
    ... items kopieren ...
}


MyArray a;
/* a befüllen ... */
MyArray b(a);
a.free_all();
```

```
class MyArray {
    Object *items;
    size_t cnt;
public:
    MyArray(); /* ... */
    MyArray(const MyArray &) = delete;
    void free_all();
};

MyArray a;
/* a befüllen ... */

MyArray b(a);    // nicht erlaubt
```

```
MyArray a;  
MyArray b = a; // Copy-Konstruktor!  
  
b = a;           // operator=
```



```
class MyArray {  
    Object *items;  
    size_t count;  
public:  
    MyArray &operator=(const MyArray &);  
};  
  
MyArray &MyArray::operator=(const MyArray &rhs) {  
    /* ... items kopieren ... */  
    return *this;  
}
```

```
{  
    MyArray a;  
    /* ... a befüllen ... */  
}
```

a.items gehen verloren!

```
class MyArray {  
    Object *items;  
    size_t count;  
public:  
    ~MyArray();  
};  
  
MyArray::~MyArray() {  
    free_all();  
}
```

## Verschieben statt Kopieren von *rvalues*

```
class Objekt {  
    ...  
};  
  
Objekt f(void) {  
    Objekt k;  
    return k;  
}  
  
int main() {  
    Objekt x = f();  
}
```

k ist temporär,  
kann nicht weiter  
verwendet werden

x existiert bis zum  
Ende von main()

x wird mit  
k initialisiert



- Wie weist man `unique_ptr` einander zu?

```
class A {  
private:  
    string s;  
public:  
    A() : s("A") {}  
    A(A&& a) : s(move(a.s)) {}  
    void show() const {  
        cout << "A: " << s << endl;  
    }  
};
```

rvalue-Referenz

```
A a;  
A b(move(a));  
a.show();  
b.show();
```

```
unique_ptr<A> a(new A);  
  
unique_ptr<A> b(move(a));
```

```
class A {  
    int a;  
public:  
    A(int value) : a(value) {}  
    A() = default;  
};
```

Default-Konstruktor  
erzeugen lassen

```
...  
A a(7);  
A b;
```

nicht erlaubt: kein Default-Konstruktor

```
class A {  
    Object *items;  
public:  
    A(const A &) = delete;  
};
```

Konstruktor  
explizit löschen

```
...  
A a;  
A b(a);
```

Copy-Konstruktor wird  
automatisch erzeugt

Default-Konstruktoren und Destruktor werden nur automatisch erzeugt, wenn weder Konstruktor noch Destruktor definiert wurden.

- Immer Klassendefinition angeben:

- nichts
- Destruktor, Copy-Konstruktor, Copy Assignment Operator
- Destruktor, Copy-Konstruktor, Copy Assignment Operator, **[Move-Konstruktor],  
Move Assignment Operator**

"Rule of Three"

"Rule of Five"