Ein Objekt und sein Scope

Jeder Objekttyp, der mittels `struct` oder `class` definiert wird, erzeugt in C++ automatisch den Scope (Namespace) seines Objektnamens, d.h. `struct Komplexe_Zahl` (oder `class Komplexe_Zahl`) erzeugt auch den Namespace `Komplexe_Zahl`. Alle Bestandteile des Typs liegen automatisch in diesem Scope. Der vollständige Name der Objekt-Bestandteile ist z.B.:

```
Komplexe_Zahl::re          // vollständiger Name des Realteil
Komplexe_Zahl::abs()       // vollständiger Name der Methode
```

Den Scope **darf man weglassen**, wenn der Zugriff über eine Referenzinstanz erfolgt, da C++ aus der Referenzinstanz den Typ und damit den Scope erkennt:

`z.re`	ist dasselbe wie	`z.Komplexe_Zahl::re`
`z.abs()`	ist dasselbe wie	`z.Komplexe_Zahl::abs()`

Wird ein Attribut (eine Methode) ohne Referenzinstanz verwendet (z.B. bei der Definition einer Methode außerhalb der Objektdefinition), **muss der Scope mit angegeben werden**:

```
double Komplexe_Zahl::abs() const  // Definition der Methode
{ return sqrt(re*re + im*im); }    // Code der Methode
```

Beachte, dass hier die Deklaration und die Definition der Methode komplett übereinstimmen müssen (insbesondere auch das **`const` und Zusätze wie `constexpr noexcept`**). **In jedem Fall muss die Methode innerhalb des Objekts zumindest deklariert werden: Nur was innerhalb der Objektdefinition steht, gehört zum Objekt.**

Falls ein Objekt zu mehr als einem Typ gehört (das geschieht bei der Vererbung: z.B. ist jeder `std::ofstream` auch ein `std::ostream`) und ein Attributname mehrmals verwendet wird, kann man den richtigen Scope angeben. Hätten beispielsweise beide ein Attribut `i`:

```
std::ofstream out;          // out ist ofstream und ostream
std::cout << out.i;                      // i des ofstream
std::cout << out.std::ostream::i;        // i des ostream
```

Das kommt nicht allzu oft vor (insbesondere haben diese 2 Typen in Wirklichkeit kein solches `i`), manchmal braucht man es aber doch (z.B. beim Überladen von virtuellen Methoden).