Praxisorientierte Einführung in C++
Lektion: "Namespaces"

Christof Elbrechter

Neuroinformatics Group, CITEC

May 28, 2014

Table of Contents

Motivation

- C++ - One Definition Rule
- In komplexer Software Namenskonflikte möglich
- Insbesondere bei Verwendung von Bibliotheken
- Guter Stil in Programmiersprachen wie C:
  - Alle Bezeichner in eigener Bibliothek mit möglichst eindeutigem Prefix versehen

```
void superlib_init();
void superlib_create_foo();
```

- Lösung in C++:
  - Namensräume

Deklarationen und Definitionen in Namensraum

- Schlüsselwort `namespace`
- Gefolgt von Block, der Deklarationen und Definitionen enthält
- Kein Semikolon am Ende

```
namespace superlib {
  int x;
  class Foo { ... };
}
```

- Methoden- und Funktions*definitionen* auch ausserhalb des Blocks möglich
  - Dann muss der Namensraum explizit genannt werden
  - Hierzu wird `::` verwendet, analog zu Klassen

```
// player.h
namespace projekt_xyz {
  class Player {
    public:
      Player();
      ~Player();
      ...
  };
}
```

```
// player.cpp
namespace projekt_xyz {
  Player::Player() { ... }
}

projekt_xzy::Player::~Player() { ... }
```

Benutzung von Bezeichnern aus Namensräumen

- Wir haben schon ein paar Beispiele gesehen

```
std::cin, std::cout, std::error, std::string
```

- Generelle Regel
  - Sei `Y` ein Bezeichner im Namensraum `X`, dann ist Bezeichner `X::Y`
- Das kann manchmal mühselig sein, daher:

```
using X::Y;
using namespace X;
```

- Beispiel:

```
using std::cout;
using namespace std;
```

Beispiel für using

```
#include "player.h"
int main() {
  projekt_xyz::Player myPlayer;
  ...
}
```

```
#include "player.h"
using projekt_xyz::Player;
int main() {
  ...
  Player myPlayer;
  ...
}
```

```
#include "player.h"
using namespace projekt_xyz;
int main() {
  ...
  Player myPlayer;
  ...
}
```

Namensauflösung

- Innerhalb eines `namespace X`-Blocks können Bezeichner aus dem Namespace ohne `X::`-Prefix benutzt werden
- Bei Konflikten wird Bezeichner aus `X` bevorzugt
  - Analog zu Klassen/Funktionen/Blöcken
- Wenn gewünscht, kann Name auch explizit per `X::` genannt werden
- Wenn Bezeichner aus anderem Namensraum `Y` gewünscht: Expliziter Prefix `Y::`

Beispiel

```
namespace projekt_xyz {
  void foo() {
    ...
  }

  void bar() {
    foo(); // Aufruf ohne Prefix projekt_xzy::
  }
}
```

Noch ein Beispiel

```
// player.h
#include <iostream>

namespace projekt_xyz {
  int cout;

  class Player {
    ...
    public: Player() {
      ...
      cout = 1;
      int projekt_xyz = 1;
      std::cout << cout;
      ...
    }
    ...
  };
}
```

Verschachtelte Namensräume

- Namensräume können verschachtelt sein

```
namespace X {
  namespace Y {
    int x;
  }
}

X::Y::x = 0;
```

Umbenennung eines Namensraums

- Namensräume, besonders verschachtelte, können unhandlich werden
- Umbennenung möglich

```
namespace new_name = old_name;
```

- Beispiel

```
namespace xyz = projekt_xyz;
namespace abc = A::B::C;
```

Anonyme Namensräume

- Anonyme Namensräume
- Beschränken Sichtbarkeit eines Bezeichners auf aktuelle Quellcode-Datei
- Geben Bezeichnern aus anonymen Namensraum Vorrang (analog zu benamsten Namensräumen)

```
// player.h
#include <iostream>
using namespace std;

namespace {
  int cout;

  foo() {
    std::cout << cout << endl;
    ...
  }
}
```

Anonyme Namensräume

- Damit können auch die Sichtbarkeit von Symbolen von lokal verwendeten Hilfklassen auf eine Compilation-Unit beschränkt werden

```
// something.cpp
namespace {
  struct MyUtil{
     MyUtil(){
        ...
     }
  };
}
void Something::foo(){
   MyUtil x;
   ...
}
```

Expliziter Zugriff auf globalen Namensraum

- Globaler Namensraum ist wie andere Namensräume, aber mit Prefix ::
- Beispiel

```
#include <iostream>

void f() {

}

namespace X {
  void f() {
    ::f();
  }
}

int main() {
  X::f();
}
```