Smart Pointer

Vorkurs C/C++, Olaf Bergmann

Pointer und Exceptions

```
void f(void) {
{
    Objekt *p = new Objekt;
    throw "Fire!";
    delete p;
}


int main() {
  try {
     f();
  } catch (...) {
     cerr << "Something went wrong!" << endl;
  }
}
```

In `f()` tritt eine Exception auf
→ Funktion wird sofort verlassen
→ p wird **nicht freigegeben**

Smart Pointer

- Pointer in C/C++ sind fehleranfällig und unflexibel
  → `unique_ptr`
- `unique_ptr<T,Deleter=default_deleter>(&`*Object*`)`
  - Alleiniger Eigentümer von *Object*
  - Zerstört *Object* automatisch, wenn nicht mehr sichtbar
  - Zerstört *Object* automatisch, wenn `unique_ptr` anderen Inhalt bekommt

Resource Acquisition is Initialization (RAII)

Smart Pointer: Benutzung

```
#include <iostream>
#include <memory>

...
std::unique_ptr<int> p(new int);

*p = 25;
std::cout << p.get() << ": " << *p << std::endl;


auto q = std::make_unique<int[]>(5);
q[3] = 37;
```

Freigeben eines Pointers

```
{
    unique_ptr<Objekt> p = make_unique<Objekt>();
    throw "Fire!"
}
```

Automatische Freigabe bei Zerstörung des `unique_ptr`-Objekts am Blockende.

```
unique_ptr<Objekt> p = make_unique<Objekt>();
p.reset();
```

`reset()` ruft Destruktor auf und gibt belegten Speicher frei

Wie übergibt man einen unique_ptr?

```
#include <memory>

...
unique_ptr<Objekt> p = make_unique<Objekt>();
```

`make_unique<Typ>(...)`: erzeugt und initialisiert Instanz von Typ, liefert Typ *

Falsch: `unique_ptr<Objekt> q = p;`
→ Copy-Konstruktor explizit verboten

```
unique_ptr<Objekt> q = move(p);
```

`std::move()` erzwingt Nutzung des Move-Konstruktors für Initialisierung, `p` wird `nullptr`

shared_ptr

- `unique_ptr`: Es gibt genau einen Verweis, Zuweisung nur per `move()`
- mehrere Verweise auf das selbe Objekt?
  → `shared_ptr`

„Reference Counting“

```
shared_ptr<Objekt> p = make_shared<Objekt>();
shared_ptr<Objekt> q = p;

p.reset();
q.reset();
```

p und q zeigen auf das selbe Objekt

Freigabe von p, q unverändert

Freigabe von q, Objekt wird zerstört

weak_ptr

- `weak_ptr`: sicherer Verweis auf `shared_ptr`
- Reference counter wird nicht erhöht