ToiletPaper #62

jambit WHERE INNOVATION WORKS

Copy- und Swap-Idiom

Von Marian Wieczorek

✖ Problem

Man möchte eine C++-Klasse schreiben, die laut „rule of three“ einen benutzerdefinierten **Destruktor**, einen **Copy-Konstruktor** und einen **Zuweisungsoperator** verlangt. Sorgen rufen hierbei **Code-Duplizierung**, **Korrektheit** und die **Exception**-Sicherheit hervor.

✔ Lösung

Das Copy- und Swap-Idiom verwendet den Copy-Konstruktor und eine Swap-Funktion, um den Zuweisungsoperator sicher zu implementieren. Ein Nachteil dabei ist, dass die Zuweisung jetzt eine Kopie erfordert. Allerdings sollte bedacht werden, spät zu optimieren!

➔ Beispiel

Ein Gegenbeispiel, das Code dupliziert, dabei Selbstzuweisung vermeiden muss und nicht Exception-sicher ist:

```
class Array {
private: Value* values; size_t size;
public:
     Array(const Array& other) : values(new Value[other.size]), size(other.size) {
          for(size_t i = 0; i < other.size; ++i) { this->values[i] = other.values[i]; }
     }
     …
     Array& operator=(const Array& rhs) {
          if (this == &rhs) { return *this; }
          delete[] this->values; // Never forget to clean up!
          this->values = new Value[rhs.size];
          for(size_t i = 0; i < rhs.size; ++i) { this->values[i] = rhs.values[i]; }
          this->size = rhs.size;
          return *this;
     }
     …
};
```

Avoid self-assignment. Otherwise, the pointer is released before it is copied.

Code duplication.

Assignment of Value might throw an exception and leave this->values in an inconsitent state (partial update).

Die Swap-Funktion, der Copy-Konstruktor und der Zuweisungsoperator des Copy- und Swap-Idioms:

```
friend void swap(Array& lhs, Array& rhs) { // One of the few good friends in C++.
     using std::swap; // Propose std::swap function as fallback to ADL.
     swap(lhs.values, rhs.values); // Only pointers are swapped.
     swap(lhs.size, rhs.size);
}

Array(const Array& other) : values(new Value(other.size)), size(other.size) {
     for(size_t i = 0; i < other.size; ++i) { this->values[i] = other.values[i]; }
}

Array& operator=(Array rhs) {
     using std::swap;
     swap(*this, rhs);
     return *this;
}
```

Pass a copy instead of a reference. Now, exceptions do not change *this*. Furthermore, self-assignment isn't an issue anymore.

No code duplication.

+ Weiterführende Aspekte

- C++ 11 – Konstruktor und Zuweisungsoperator verschieben („rule of five“)