

Klausurvorbereitung

Intelligente Zeiger

Clicker: Smart-Pointer Kopieren (4)

```
void f1(unique_ptr<Z>& ap){
    ap->use();
}
void CopyTestAuto2() {
    unique_ptr<Z> up(new Z(11));
    up->use();
    f1(up);
    // up->use(); // kein Problem
    cout << "Ende ";
}
```

Ausgabe?

1. +Z11 use:11 use:11 Ende -Z11
2. +Z11 use:11 use:11 Ende
3. +Z11 use:11 use:11 -Z11 Ende
4. +Z11 Ende

Clicker: Smart-Pointer Kopieren (4)

```
void f1(unique_ptr<Z>& ap){
    ap->use();
}
void CopyTestAuto2() {
    unique_ptr<Z> up(new Z(11));
    up->use();
    f1(up);
    // up->use(); // kein Problem
    cout << "Ende ";
}
```

Ausgabe?

1. +Z11 use:11 use:11 Ende -Z11
2. +Z11 use:11 use:11 Ende
3. +Z11 use:11 use:11 -Z11 Ende
4. +Z11 Ende

```
+Z11 use:11 use:11 Ende -Z11 // +Z11 use:11 use:11 Ende
// +Z11 use:11 use:11 -Z11 Ende// +Z11 Ende
```

Clicker: `unique_ptr` haben Verschiebe-Konstruktor

```
void f5(unique_ptr<Z> ap){
    ap->use();
}
void CopyTestAuto5() {
    unique_ptr<Z> up(new Z(11));
    up->use();

    // f5(up); // Compilerfehler

    f5( ::move(up) ); // wandelt in R-Value (temp. Variable um)
    cout << "Ende ";

    //up->use(); / wäre nun ein Fehler up ungültig
}
```

Clicker: unique_ptr haben Verschiebe-Konstruktor (2a)

```
void f6(unique_ptr<Z> ap) // als Wert
{
    cout << "f6 ";
}
void CopyTestAuto6() {
    unique_ptr<Z> up(new Z(12));
    f6(move(up));
    cout << "Ende ";
}
```

Ausgabe?

1. +Z12 +Z4 -Z12 f6 Ende -Z4
2. +Z12 +Z12 f6 Ende
3. +Z12 f6 -Z12 Ende
4. +Z12 f6 Ende -Z12
5. +Z12 +Z4 f6 -Z4 Ende

Clicker: unique_ptr haben Verschiebe-Konstruktor (2a)

```
void f6(unique_ptr<Z> ap) // als Wert
{
  cout << "f6 ";
}
void CopyTestAuto6() {
  unique_ptr<Z> up(new Z(12));
  f6(move(up));
  cout << "Ende ";
}
```

Ausgabe?

1. +Z12 +Z4 -Z12 f6 Ende -Z4
2. +Z12 +Z12 f6 Ende
3. +Z12 f6 -Z12 Ende
4. +Z12 f6 Ende -Z12
5. +Z12 +Z4 f6 -Z4 Ende

1. +Z12 +Z4 -Z12 f6 Ende -Z4
2. +Z12 +Z12 f6 Ende
- 3. +Z12 f6 -Z12 Ende**
4. +Z12 f6 Ende -Z12
5. +Z12 +Z4 f6 -Z4 Ende

Clicker: unique_ptr haben Verschiebe-Konstruktor (2b)

```
void f6(unique_ptr<Z> ap) // als Wert
{
    Z* hlp = new Z(4);
    ap = unique_ptr<Z>(hlp);
    cout << "f6 ";
}
void CopyTestAuto6() {
    unique_ptr<Z> up(new Z(12));
    f6(move(up));
    cout << "Ende ";
}
```

Ausgabe?

1. +Z12 +Z4 -Z12 f6 Ende -Z4
2. +Z12 +Z4 f6 -Z12 -Z4 Ende
3. +Z12 +Z4 -Z12 f6 -Z4 Ende
4. +Z12 +Z4 -Z12 -Z4 f6 Ende
5. +Z12 +Z4 f6 -Z4 Ende

Clicker: unique_ptr haben Verschiebe-Konstruktor (2b)

```
void f6(unique_ptr<Z> ap) // als Wert
{
    Z* hlp = new Z(4);
    ap = unique_ptr<Z>(hlp);
    cout << "f6 ";
}
void CopyTestAuto6() {
    unique_ptr<Z> up(new Z(12));
    f6(move(up));
    cout << "Ende ";
}
```

Ausgabe?

1. +Z12 +Z4 -Z12 f6 Ende -Z4
2. +Z12 +Z4 f6 -Z12 -Z4 Ende
3. +Z12 +Z4 -Z12 f6 -Z4 Ende
4. +Z12 +Z4 -Z12 -Z4 f6 Ende
5. +Z12 +Z4 f6 -Z4 Ende

1. +Z12 +Z4 -Z12 f6 Ende -Z4
2. +Z12 +Z4 f6 -Z12 -Z4 Ende
3. +Z12 +Z4 -Z12 f6 -Z4 Ende
4. +Z12 +Z4 -Z12 -Z4 f6 Ende
5. +Z12 +Z4 f6 -Z4 Ende

Clicker: unique_ptr mit Methode release

```
void CopyTestAuto9() {  
    unique_ptr<Z> up(new Z(12));  
    Z* p = up.get();  
    up.release(); // gibt Speicher frei; setzt internen Zeiger auf nullptr.  
    p->use();  
    up->use();  
    cout << "Ende ";  
}
```

Ausgabe?

1. +Z12 use:12 use:12 Ende -Z12
2. +Z12 **Bumm**
3. +Z12 use:12 **Bumm**
4. +Z12 use:12 use:12 Ende **Bumm** -Z12
5. Compilerfehler

Clicker: unique_ptr mit Methode release

```
void CopyTestAuto9() {  
    unique_ptr<Z> up(new Z(12));  
    Z* p = up.get();  
    up.release(); // gibt Speicher frei; setzt internen Zeiger auf nullptr.  
    p->use();  
    up->use();  
    cout << "Ende ";  
}
```

Ausgabe?

1. +Z12 use:12 use:12 Ende -Z12
2. +Z12 **Bumm**
3. +Z12 use:12 **Bumm**
4. +Z12 use:12 use:12 Ende **Bumm** -Z12
5. Compilerfehler

1. +Z12 use:12 use:12 Ende -Z12
2. +Z12 Bumm
3. +Z12 use:12 **Bumm**
4. +Z12 use:12 use:12 Ende **Bumm** -Z12
5. Compilerfehler

Clicker: unique_ptr als Rückgabewerte

```
unique_ptr<Z> f10(){
    unique_ptr<Z> up(new Z(4));
    cout << "f10 ";
    return up;
}
void CopyTestAuto10() {
    unique_ptr<Z> up = f10();
    cout << "Ende ";
}
```

Ausgabe?

1. +Z4 f10 Ende -Z4
2. +Z4 10 **Bumm**
3. +Z4 f10 +Z4 Ende -Z4 -Z4
4. +Z4 f10 Ende -Z4 -Z4

Clicker: unique_ptr als Rückgabewerte

```
unique_ptr<Z> f10(){
    unique_ptr<Z> up(new Z(4));
    cout << "f10 ";
    return up;
}
void CopyTestAuto10() {
    unique_ptr<Z> up = f10();
    cout << "Ende ";
}
```

Ausgabe?

1. +Z4 f10 Ende -Z4
2. +Z4 10 **Bumm**
3. +Z4 f10 +Z4 Ende -Z4 -Z4
4. +Z4 f10 Ende -Z4 -Z4

1. +Z4 f10 Ende -Z4 // Move-Semantik und ein neues Z wird sowieso nicht erzeugt
2. +Z4 10 **Bumm**
3. +Z4 f10 +Z4 Ende -Z4 -Z4
4. +Z4 f10 Ende -Z4 -Z4

Clicker: shared_ptr und Rekursion (1)

```
class R{
public:
    R(int id = 9) : m_id(id) {
        cout << "+R" << id << " "; }
    ~R() {cout<<"-R"<< m_id << " "; }
    void addFr(shared_ptr<R> f){
        myFr.push_back(f); }
private:
    vector<shared_ptr<R>> myFr;
    int m_id;
};
```

```
void friendTest1(){
    shared_ptr<R> mike(new R(1));
    shared_ptr<R> anne(new R(2));
    mike->addFr(anne);
    if (2 < 3){
        shared_ptr<R> bert(new R(3));
        anne->addFr(bert);
    }
}
```

Ausgabe?

1. +R1 +R2 +R3 -R1 -R2 -R3
2. +R1 +R2 +R3 -R3 -R2 -R1
3. +R1 +R2 +R3 -R1 -R3 -R2
4. +R1 +R2 +R3 -R2 -R3 -R1

Clicker: shared_ptr und Rekursion (1)

```
class R{
public:
    R(int id = 9) : m_id(id) {
        cout << "+R" << id << " "; }
    ~R() {cout<<"-R"<< m_id << " "; }
    void addFr(shared_ptr<R> f){
        myFr.push_back(f); }
private:
    vector<shared_ptr<R>> myFr;
    int m_id;
};
```

1. +R1 +R2 +R3 -R1 -R2 -R3

Bert wird freigegeben, noch bei Anne benutzt,
Anne wird freigegeben, noch bei Mike benutzt.
Mike wird freigegeben, damit wird Anne
bei Mike richtig freigegeben und damit
auch Bert richtig über Anne

```
void friendTest1(){
    shared_ptr<R> mike(new R(1));
    shared_ptr<R> anne(new R(2));
    mike->addFr(anne);
    if (2 < 3){
        shared_ptr<R> bert(new R(3));
        anne->addFr(bert);
    }
}
```

Ausgabe?

1. +R1 +R2 +R3 -R1 -R2 -R3
2. +R1 +R2 +R3 -R3 -R2 -R1
3. +R1 +R2 +R3 -R1 -R3 -R2
4. +R1 +R2 +R3 -R2 -R3 -R1