Übungsblatt 8

Wichtig: Abgabe des dritten Meilensteins

Bitte denken Sie daran, Ihre Implementierung des dritten Meilensteins bis spätestens **17.12.** fertigzustellen und die entsprechend markierte Revision auf den `git`-Server des Lehrstuhls zu pushen.

Aufgabe 8.1: Übersetzung von Klassen – Einfachvererbung

Gegeben sei folgender Code (Syntax und Semantik wie aus der Vorlesung bekannt):

```
class O {
  int v1;
  void foo() { print("O::foo"); }
  void bar() { print("O::bar"); }
}

class U: O {
  int v2;
  void bar() { print("U::bar"); }
  void buz() { print("U::buz"); }
}
```

```
void callO(O o) {
  int x = o.v1;
  o.bar();
}

void callU(U u) {
  int x = u.v1 + u.v2;
  u.bar();
}
```

Erläutern Sie schrittweise, wie dieses Beispiel übersetzt werden kann:

a) Was versteht man unter *statischen Typen* und *dynamischen Typen*?

b) Wie wird der Zugriff auf die Instanzvariablen `v1` bzw. `v2` übersetzt? Wie sehen Objekte des Typs `O` bzw. `U` im Speicher aus? bzw. `v2` übersetzt?

c) Nehmen Sie zunächst an, dass Methodenaufrufe *nicht-virtuell* (*non-virtual*) sind. Was bedeutet dies für die Methodenaufrufe in `callO()` bzw. `callU()`? Wie werden solche nicht-virtuellen Aufrufe übersetzt?

d) Nehmen Sie nun an, dass Methodenaufrufe *virtuell* (*virtual*) sind. Was ändert sich nun für die Aufrufe in `callO()` bzw. `callU()`? Welches Optimierungspotential besteht bei virtuellen Aufrufen?

Templates für 8.1:

8.1.a – Template 1 — Schreiben Sie Code bei dem der dyn. Typ nicht statisch bestimmbar ist

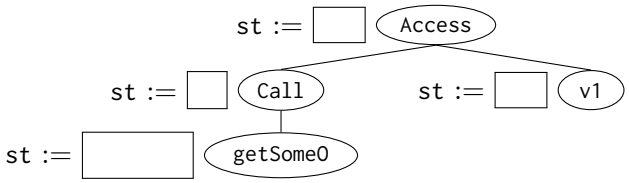
```
O o;









o.bar();
```

8.1.a – Template 2 — Schreiben Sie Code der einen Typfehler zur Laufzeit provoziert

```
void makeTypeError(                          ) {









}

makeTypeError(new                          );
```

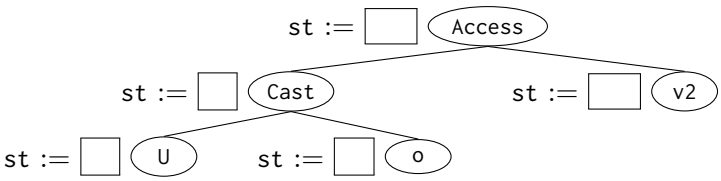
8.1.a – Template 3 — Bestimmen Sie die statischen Typen

```
O giveSomeO() { ... }

giveSomeO().v1;
```



8.1.a – Template 4 — Bestimmen Sie die statischen Typen

```
O o;
...
((U) o).v2;
```



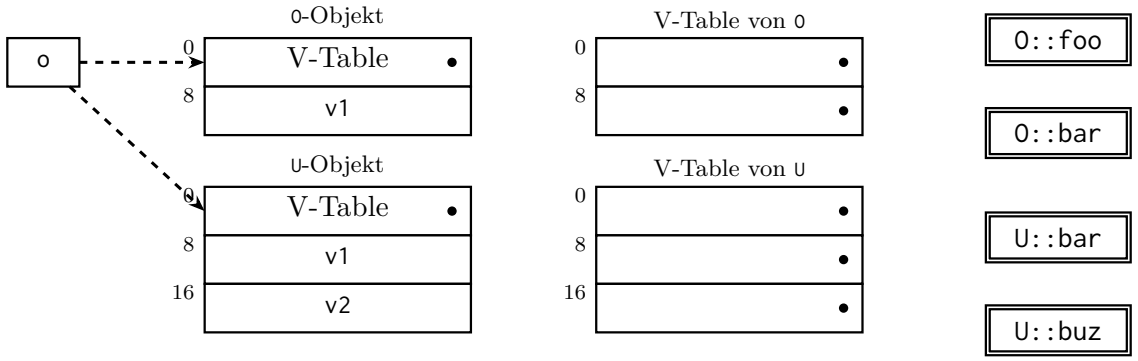
8.1.c – Template 5 — Ergänzen Sie!

```
O o = new U();
U u = new U();
o.bar();         // statischer Typ ist [         ], Aufruf von [         ]
u.bar();         // statischer Typ ist [         ], Aufruf von [         ]
((O) u).bar();   // statischer Typ ist [         ], Aufruf von [         ]
((U) o).bar();   // statischer Typ ist [         ], Aufruf von [         ]
```

8.1.d – Template 6 — Ergänzen Sie!

```
O o = new U();
U u = new U();
o.bar();         // dynamischer Typ ist [         ], Aufruf von [         ]
u.bar();         // dynamischer Typ ist [         ], Aufruf von [         ]
((O) u).bar();   // dynamischer Typ ist [         ], Aufruf von [         ]
((U) o).bar();   // dynamischer Typ ist [         ], Aufruf von [         ]
```

8.1.d – Template 7 — Ergänzen und verpointern Sie das Objektlayout!



Aufgabe 8.2: Laufzeittypprüfungen – Einfachvererbung

Es sei nun folgender Code gegeben:

```
class A { ... }

class B: A { ... }

class C: B { ... }

class D: A { ... }
```

```
void castToA(B b) {
  A a = (A) b;
}

void castToB(A a) {
  B b = (B) a;
}
```

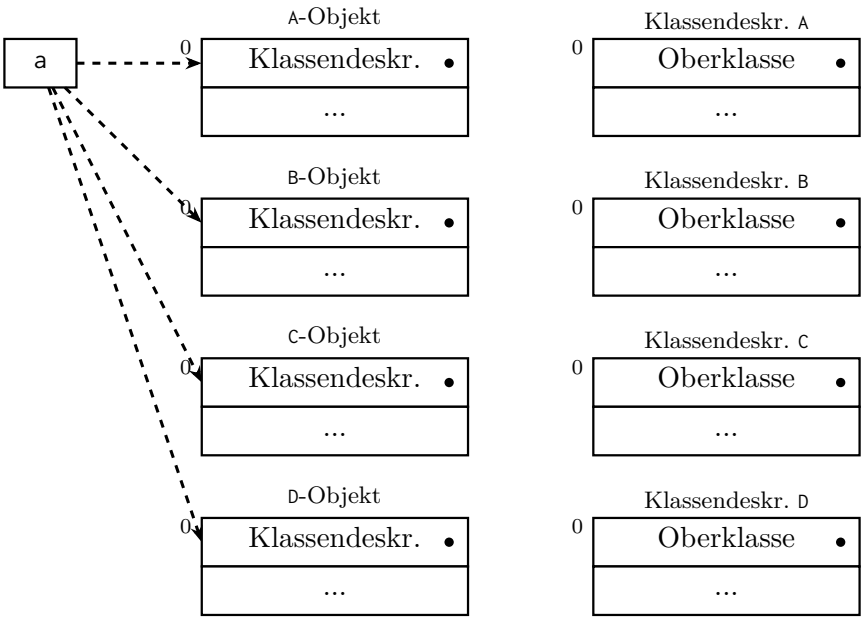
Erläutern Sie, wie der gezeigte Code übersetzt werden kann:

a) Was sind Klassendeskriptoren und wofür werden diese benötigt?

b) Was passiert bei den gezeigten *Typwandlungen* und *Typprüfungen* zur Übersetzungs- bzw. Laufzeit? Erläutern Sie sowohl die Variante *ohne* als auch die Variante *mit* Displays.

Templates für 8.2:

8.2.a – Template 1 — Verpointern Sie



8.2.b – Template 2 – Implementieren Sie `instanceof` mittels Pointerverfolgung

```
// !!ClassDescriptor has field super_class!!
boolean _instanceof(Object o, ClassDescriptor t) {
  ClassDescriptor c = o.class_descriptor;
  while                          // TODO









}
```

Aufgabe 8.3: Übersetzung von Interfaces

Gegeben sei der folgende Code mit Interfaces:

```
interface I {
  void foo();
}

class A: I {
  int v1;
  void foo() { print(self.v1); }
  void bar() { print("A::bar"); }
}

class B: I {
  int v2;
  int v3;
  void buz() { print("B::buz"); }
  void foo() { print(self.v2); }
}
```

```
void callI(I i) {
  i.foo();
}

void castToI(A a) {
  I i = (I) a;
}

void castToA(I i) {
  A a = (A) i;
}
```

Erläutern Sie, wie der gezeigte Code übersetzt werden kann:

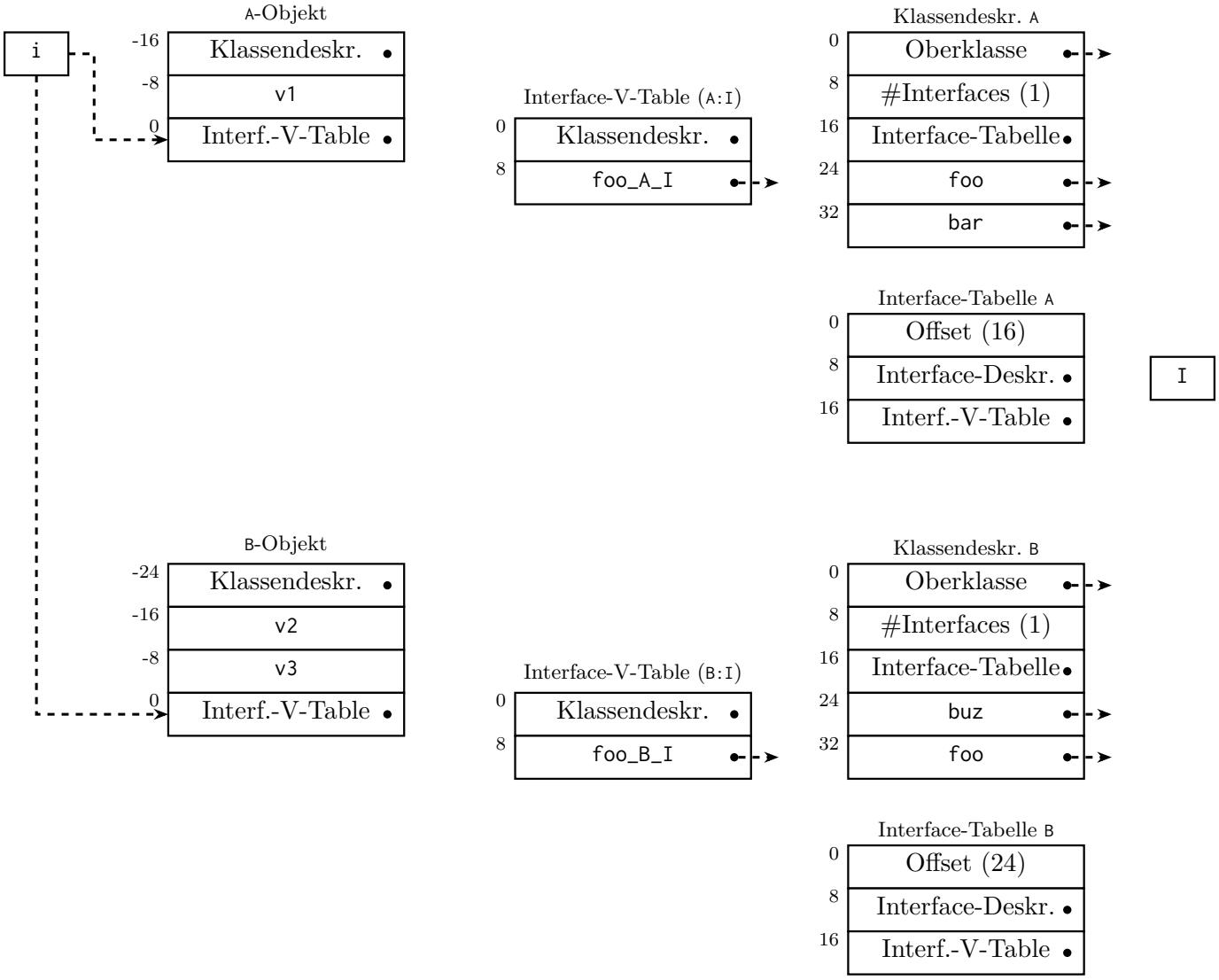
a) Warum kann bei der Verwendung von Interfaces nicht derselbe Ansatz wie bei Einfachvererbung verwendet werden? Wie werden Interfaces stattdessen realisiert?

b) Was passiert bei dem Methodenaufruf von `foo()` in `callI()`?

c) **Bonus, falls noch Zeit...**: Wie werden die Typwandlungen in `castToI()` bzw. `castToA()` realisiert? Was passiert jeweils zur Übersetzungszeit, was zur Laufzeit?

Templates für 8.3:

8.3.a – Verpointern Sie das Objektlayout!


A-Objekt
i
-16
Klassendeskr.
-8
v1
0
Interf.-V-Table
Interface-V-Table (A:I)
0
Klassendeskr.
8
foo_A_I
Klassendeskr. A
0
Oberklasse
8
#Interfaces (1)
16
Interface-Tabelle
24
foo
32
bar
Interface-Tabelle A
0
Offset (16)
8
Interface-Deskr.
16
Interf.-V-Table
I
B-Objekt
-24
Klassendeskr.
-16
v2
-8
v3
0
Interf.-V-Table
Interface-V-Table (B:I)
0
Klassendeskr.
8
foo_B_I
Klassendeskr. B
0
Oberklasse
8
#Interfaces (1)
16
Interface-Tabelle
24
buz
32
foo
Interface-Tabelle B
0
Offset (24)
8
Interface-Deskr.
16
Interf.-V-Table