# Verhaltenskonformanz

Für Klasseninvarianten (gelten vor und nach jedem Methodenaufruf) gilt:

Unterkl.-Invariante gilt stärker als Oberkl.-Invariante:

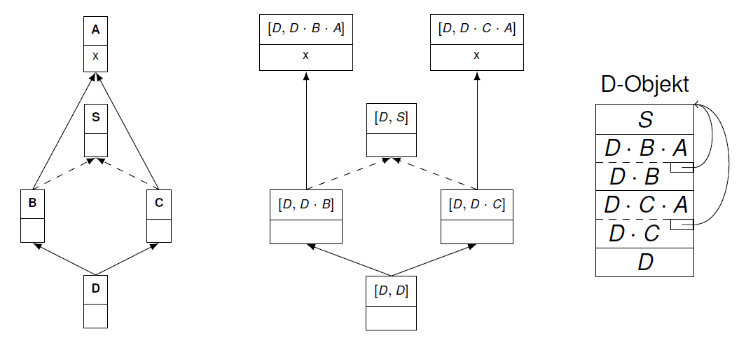
Oberklasse hat stärkere Vorbedingung (Oberklasse verlangt mehr):

Unterklasse hat stärkere Nachbedingung (Unterklasse leistet mehr):

Tipp: Bei Prüfung Bedingungen komplett ausschreiben.

# V-Tables

VTables enthalten alle Methoden, die in der jeweiligen Klasse deklariert sind!



## Dynamische Bindung

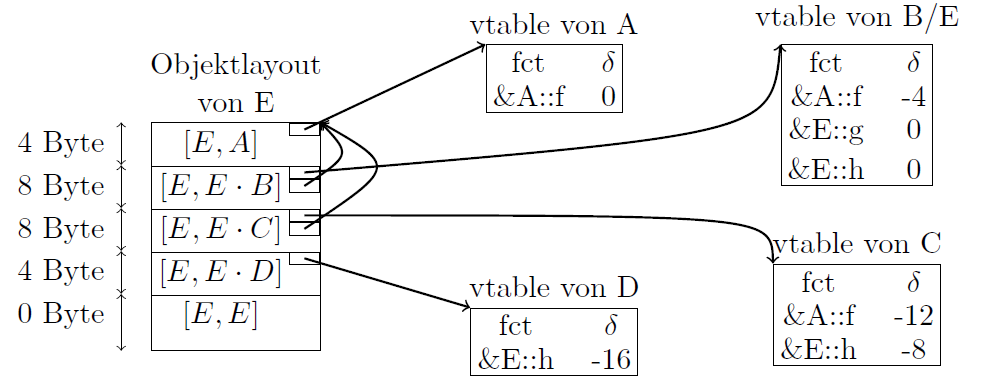
für „most-derived-class“-Fkt. .

ist die Menge aller Subobjekte, die im Subobjektgraph von aus erreichbar sind und eine Definition vom Member m enthalten.

Wenn (also alle Subobjekte in S über erreichbar sind, „Subobjekt dominiert Subobjekt-Menge S“) und , dann ist .

TODO

### Deltas



Beim Delta-Abzählen: Verwende obere Kante eines Subobjekten als Zählreferenz. Achtung: Members von betrachteter Klasse werden meist nicht in dessen Subobjekt, sondern einer geteilten vtable gespeichert.

Geteilte Vtable entlang linkester Außenkante im Subobjektgraphen.

### Umsetzung von CPP Aufrufen

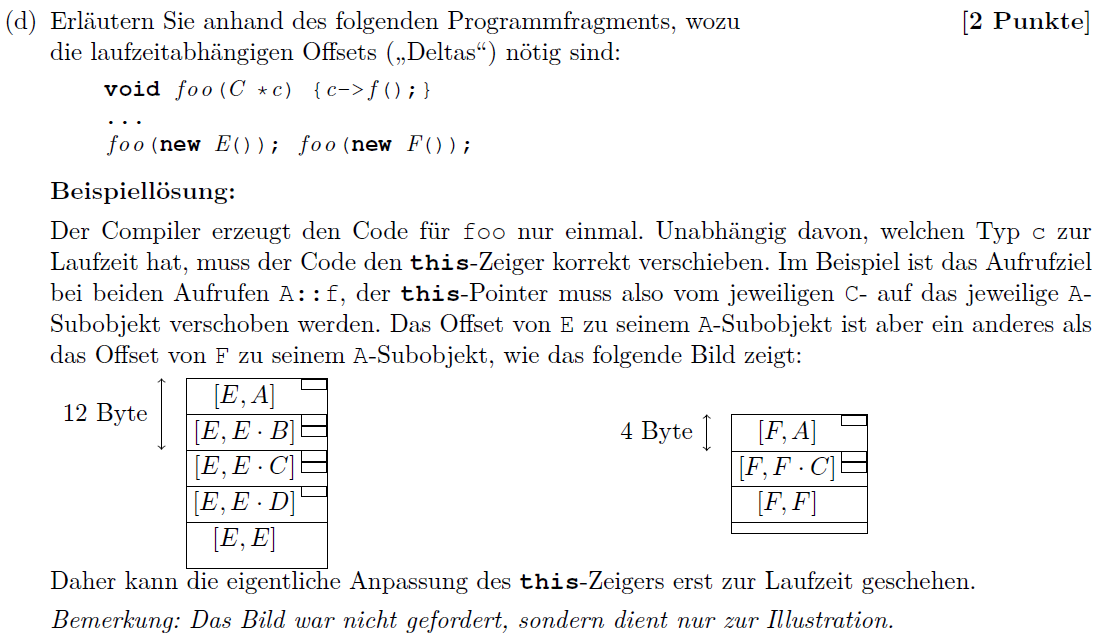
pbc->f(42);

wird umgesetzt durch:

register vt = &(pbc->vptr[K]);

(\*vt->fct)(pbc + (vt->delta), 42);

Mit K als Index der Methode in der VTable. Parameter kann auch weggelassen werden.



### Umsetzung von Upcasts

#### Nicht-virtuelle Mehrfachvererbung

B pb = pc; // pc is type C; wird zu:

B pb = (B\*) (((char\*)pc) + delta(C,B));

ist statisch zur Kompilierzeit bekannt.

#### Virtuelle Mehrfachvererbung

B pb = pc; // pc is type C; wird zu:

B pb = \*((B\*\*) (((char\*)pc)+offset(B\_ptr)));

ist der Offset des B-Subobjektzeigers im C-Objkt.

# Generics

? extends C im Typparameter: Der Typ liegt unterhalb C. Ich kann davon nur als C oder höher lesen und Object-Typen darin schreiben. Schreiben von C-Typen ist nicht möglich.

? super C im Typparameter: Der Typ liegt oberhalb C. Ich kann davon nur als Object lesen, nicht als C, dafür C-Objekte oder höher darin schreiben.

## Klausurtypische Aufgabe

todo

Nicht akzeptierte Beispiele – Aufgabe: 1-3 Punkte => eine zu restriktive Deklaration, 3-5 Punkte => zwei zu restriktive Deklarationen.

## PECS Example

Note how the source list src (the producing list) uses extends, and the destination list dest (the consuming list) uses super:

public class Collections {

public static <T> void copy(List<? super T> dest,

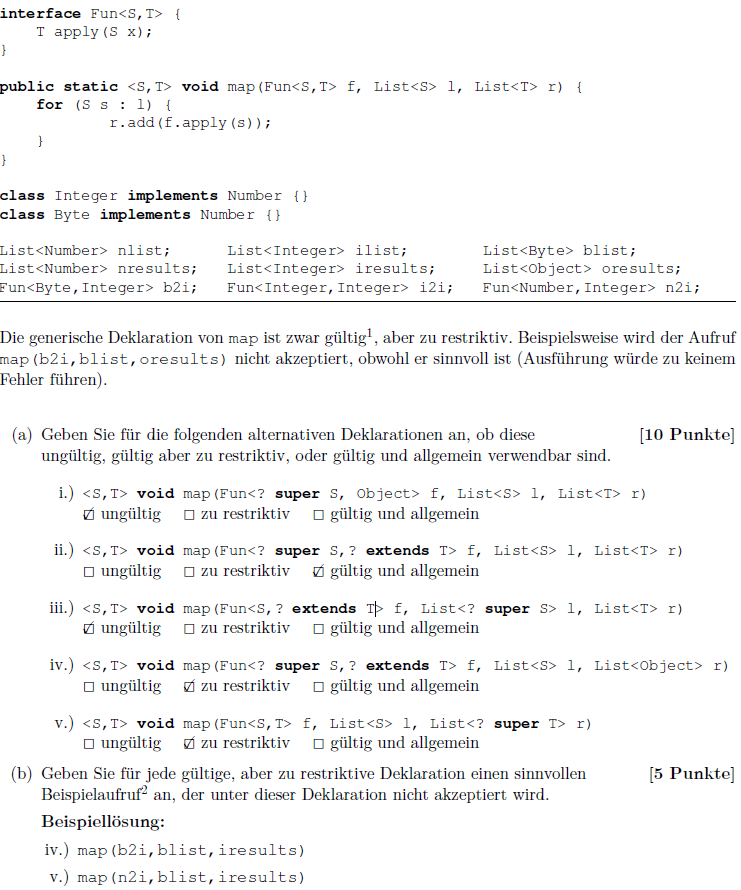
List<? extends T> src) {

for (int i = 0; i < src.size(); i++)

dest.set(i, src.get(i));

}

}



# Typsystem

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  | Allokation einer Speicherzelle. |
|  | Lesen einer Speicherzelle. |
|  | Schreiben auf eine Speicherzelle. |
| *Typregel für Objekte als Records* | |
|  | Typregel für Objekte als Records |
| *Objektkonversion* | |
| *Objekterweiterung* | |
|  | Sumsumption, Typkonformanz. |
| Subtyp-Beziehungen auf Funktionen. Kontravarianz im Parameter, Kovarianz im Ergebnis  Dh. Parameter erben in die gegensätzliche Richtung, Rückgabetypen in dieselbe! | |
|  | Reflexiv. ZB für Trivialumformung {..}=Klasse |
|  |  |
|  |  |
| Also: Polymorphe Typen erben, wenn die Typparameter in dieselbe Richtung erben. Zwischen monomorphe und polymorphen Typen kann keine Erben-Beziehung bestehen. | |

# Programmanalyse

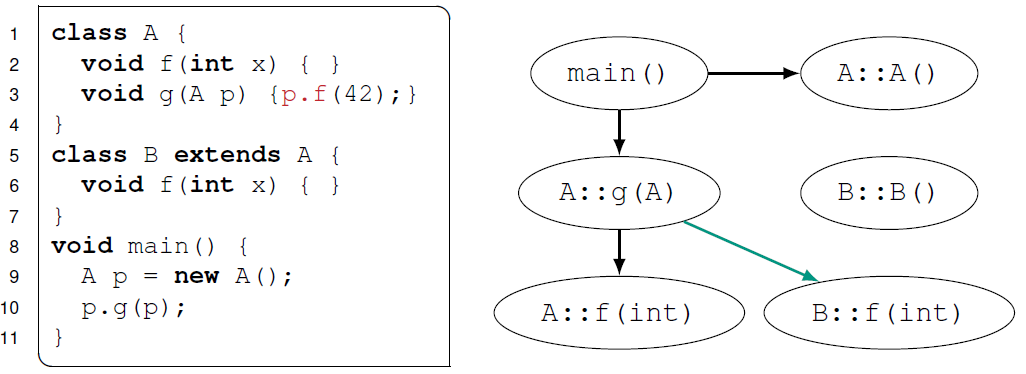
## Points-to-Analyse nach Andsersen

Fluss-sensitiv und Kontext-sensitiv.

|  |  |
| --- | --- |
|  |  |
| Zuweisung neues Objekt  O x = new O(); |  |
| Zuweisung Variable  p = q; |  |
| Return einer Funktion  class O {  R func() {return x;}} |  |
| Aufruf dynamische Funktion q=o.m(a)  mit statischem Lookup von typkorrekten o\_i! | |
| Attribut Lesen  x = p.f; |  |
| Attribut Schreiben  p.f = y; |  |

## Rapid-Type Analysis

Analysiere Aufrufstruktur als Graphen. Knoten sind Methodensignaturen, Kanten geben an ob innerhalb einer Methodensignatur eine andere aufgerufen werden kann. Wegen dynamischer Bindung müssen alle potentielle Ziele berücksichtigt werden.



Umgang mit dynamischer Bindung: Wegen konservativer Approximation, Bestimme Menge Z der potentiellen Aufrufziele: finde aufgerufene Methode durch Static Lookup (Aufwärtssuche), füge aufgerufene Funktion und alle Methoden mit passender Signatur abwärts der Hierarchie dazu.

Reduktion der Call-Größe: Siehe Zusammenfassung.

### RTA als Constraint Problem

Finde Mengen und mit: