17. POLYMORPHIE

In diesem Block wollen wir uns mit dem Unterschied zwischen statischem und dynamischem Typ sowie deren Bestimmung beschäftigen. Dieses Verständnis ist die Grundlage und Handwerkszeug für Polymorphie. Polymorphie ist ein Konzept der objektorientierten Programmierung, bei dem es im Grunde darum geht, dass der Datentyp einer Variable nicht eindeutig (sondern dynamisch) bestimmt ist, d. h. eine Variable Objekte unterschiedlichen Typs speichern kann. „Schuld" daran ist Vererbung (bzw. die damit einhergehende Überschreibung) und Überladung. Eine Methode kann in einer Unterklasse überschrieben werden (gleiche Signatur). Gleichzeitig kann eine Methode überladen sein (gleicher Methodenname, unterschiedliche Parametertypen). Die Signatur der aufgerufenen Methode ist statisch eindeutig bestimmt (durch die statischen Typen), nicht aber die tatsächlich aufgerufene Methode (da ggf. überschrieben).

I. **Bestimmung des *statischen Typs*** einer Variable:

Der statische Typ einer Variable kann einfach an der *Deklaration* abgelesen werden, d. h. er steht immer links neben dem ersten Vorkommen des Variablennamens (im Gültigkeitsbereich der Variable). Er kann nicht geändert werden (nur vorübergehend durch Casten), denn jede Deklaration ist eindeutig.

Sonderfall: Der stat. Typ von `this` entspricht der Klasse, in der es benutzt wird.

Besonderheit: *„Casting"*

Liegt ein Cast (bspw. `(A)b`) vor, so ändert sich der stat. Typ *vorübergehend* (zu `A`). Wir müssen jedoch zuerst überprüfen, ob der Cast überhaupt funktioniert:

1. *Compiler-Sicht*: Steht die Klasse des *stat. Typs* der Variable, die gecastet wird, in einer Relation (d. h. Vererbungshierarchie) mit der *„Cast"-Klasse*? Wenn die beiden Klassen etwas miteinander zu tun haben, also eine von der anderen erbt (ggf. indirekt) oder die Klassen identisch sind, dann könnte der Cast theoretisch möglich sein. Anderenfalls gibt es einen ***Compiler-Fehler***.
2. *Dynamische Sicht*: Ist das Objekt der Variable, die wir gerade versuchen zu casten, auch tatsächlich vom Typ des „Casts" (oder spezieller)? Wir sehen uns jetzt also den *dyn. Typ* der Variable an (siehe unten) und prüfen, ob diese Klasse (des dyn. Typs) von der „Cast"-Klasse erbt (ggf. indirekt) (oder die Klassen identisch sind). Ist der Typ des Objekts *nicht* mindestens so speziell wie der angegebene „Cast-Typ", so gibt es einen ***Laufzeitfehler*** (`ClassCastException`).

II. **Bestimmung des *dynamischen Typs*** einer Variable:

Den tatsächlichen (dynamischen) Typ einer Variable ermittelt man durch *„Zurückverfolgen"* bis zur Objekterzeugung. Wird der Variable `b` bspw. `a` zugewiesen (`b = a`), müssen wir nachsehen, auf welches Objekt `a` *zu diesem Zeitpunkt* zeigt. Ist die letzte Zuweisung an `a` bspw. `a = new B()`, so zeigt `b` auf genau dieses `B`, sprich `b` hat den dyn. Typ `B`. Hier interessieren uns also nicht die Deklarationen, sondern die tatsächlichen Zuweisungen. Wie der Name „dynamisch" schon sagt ist der dyn. Typ abhängig von der letzten Zuweisung, d. h. er ist veränderlich und nur zur Laufzeit des Programms bestimmbar.

III. **Bestimmung der aufgerufenen Methode** / Membervar. für Aufrufe der Form $e_0.method(e_1, \dots, e_n)$ bzw. $e_0.member$, wobei e_i Variablen (oder Konstanten) sind:

❶ ***Compiler-Sicht*** *(bei allen Methoden; gilt auch für Membervariablen)*

1. Bestimme den statischen Typ der Variable (e_0), auf dem die Methode aufgerufen wird (oben). Ggf. gibt es einen Fehler durch einen Cast.
2. Bestimme (falls nötig) auch die statischen Typen aller Parameter ($e_{1\dots n}$).
3. Wir suchen nun
   - in der in Schritt 1 gefundenen Klasse (oder einer Oberklasse davon)
   - eine Methode mit passendem Namen (wir nehmen logischerweise nicht `k(...)` wenn wir `m(...)` suchen), die
   - die Parametertypen aus Schritt 2 akzeptiert. Beachte, dass man einer Methode, die als Parameter z. B. ein `A` erwartet, auch alle Typen (Klassen) übergeben kann, die von `A` erben.

   Gibt es mehrere passende Methoden (bspw. `m(Object)`, `m(A)` und `m(B)`, wenn wir ein `B` übergeben und `B` von `A` erbt), so nehmen wir die speziellere der beiden Methoden (hier `m(B)`). Wo ein `Object` als Parameter erwartet wird kann man alles übergeben, schließlich erbt jede Klasse von `Object`.

   Achtung: Die speziellste Methode könnte auch in der Oberklasse stehen, schließlich erben wir alle Methoden der Oberklasse! Wir würden im obigen Beispiel also auch dann `m(B)` wählen, wenn diese in der Oberklasse definiert worden wäre. Beachte wiederum, dass wir keinen Zugriff auf private Methoden einer Oberklasse haben.

   Keine passende Methode gefunden? → Compiler-Fehler.

❷ ***Dynamische Sicht*** *(nur bei nicht-statischen Methoden)*

Wir führen direkt die in Schritt ❶ gefundene Methode aus, wenn diese `final` (nicht überschreibbar) oder `static` (sowieso nur statischer Kontext) ist oder wir auf dem Schlüsselwort `super` operieren (d. h. nicht wieder in der Unterklasse kucken wollen). Auch beim Zugriff auf eine Membervariable (statt Methode) entfällt Schritt ❷ (da Variablen nicht überschrieben werden).

1. Bestimme den dynamischen Typ der Variable (e_0), auf dem die Methode aufgerufen wird (oben). Sind dyn. und stat. Typ identisch, so kann man aufhören und die Methode aus Schritt ❶ ausführen. Anderenfalls:
2. Schau nun in der in Schritt 1 gefundenen Klasse (dyn. Typ) nach, ob die in Schritt ❶ gefundene Methode überschrieben wird. Dazu muss der Name der Methode und alle Parametertypen (d. h. die Signatur) *exakt* übereinstimmen! Ist das der Fall, so führen wir diese Methode aus, anderenfalls die Methode aus Schritt ❶.

   Achtung: Auch hier könnten wir die überschriebene Methode geerbt haben; nämlich von einer übergeordneten Klasse, die der Klasse des stat. Typs wiederum untergeordnet ist.