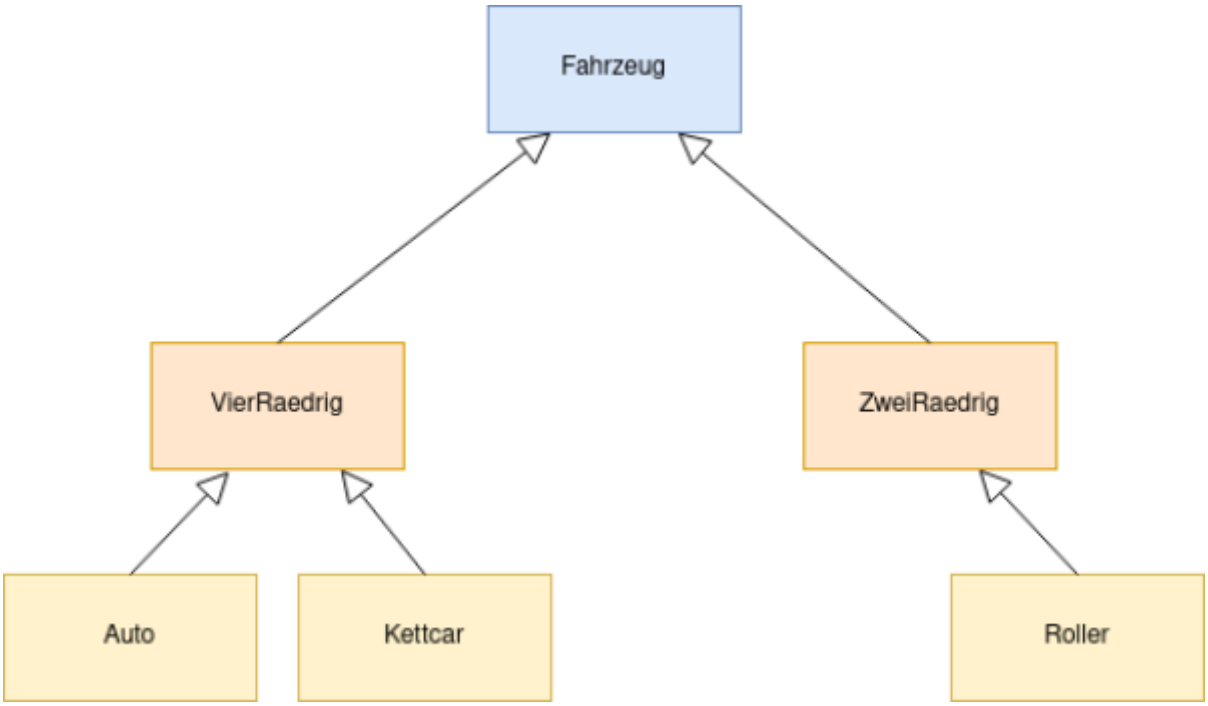
Polymorphismus genauer

Variablenpolymorphismus

Polymorphismus haben wir schon für Variablen kennengelernt: Eine Variable eines Supertyps kann auch Werte aller Subtypen halten - die Variable ist *polymorph*.



(A1)

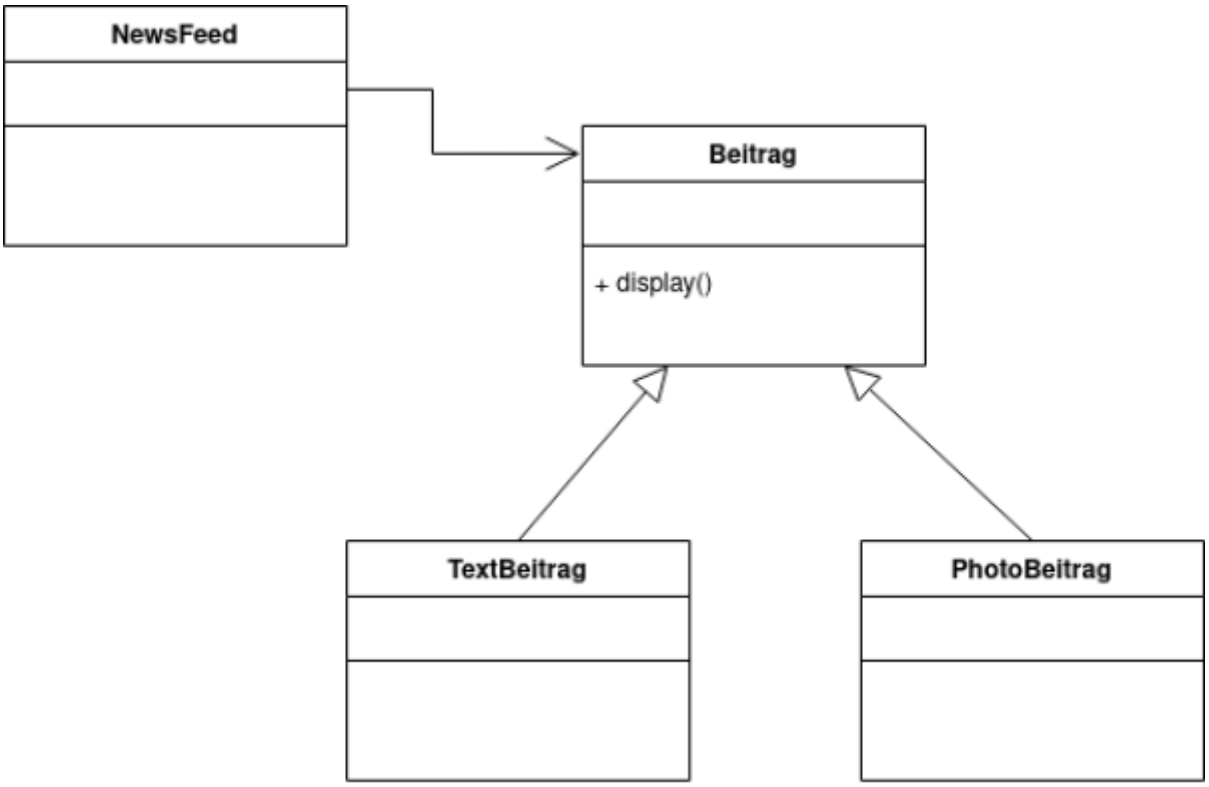
Welche Typen können Werte haben, die in den folgenden Variablen gespeichert werden?

```
Fahrzeug f;
Roller r;
vierRaedrig v;
```

Methodenpolymorphismus

Problemstellung

Die Vererbungshierarchie unseres soziales Netzwerk mit Vererbung sieht gerade so aus:



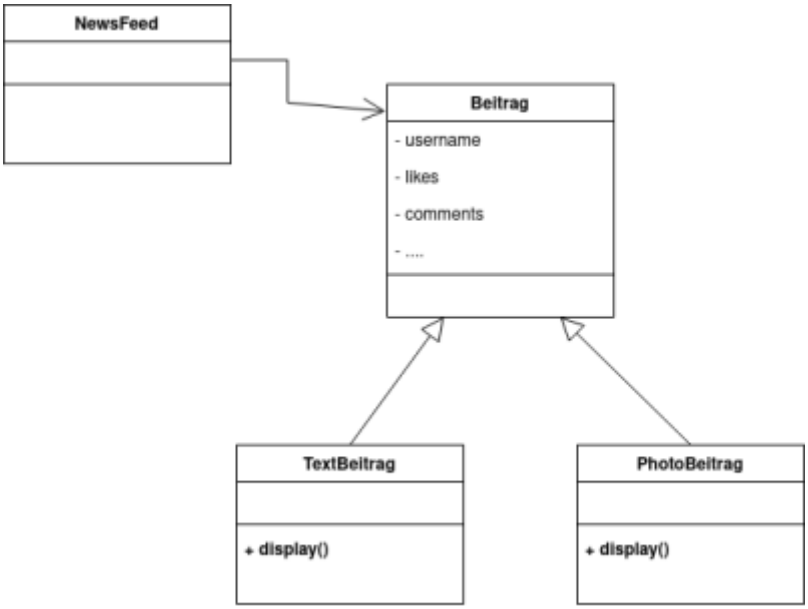
Man sieht, dass die Methode zum Anzeigen eines Beitrags in der Klasse `Beitrag` definiert ist und an die Klassen `TextBeitrag` und `PhotoBeitrag` vererbt wird. Diese Methode weiß nichts über besondere Eigenschaften der Subklassen - Vererbung ist eine Einbahnstrasse. Das führt zum Problem, dass die Ausgabe aller Beiträge etwas so aussehen:

```
Leonardo da Vinci
40 seconds ago  -  2 people like this.        TextBeitrag
  No comments.

Alexander Graham Bell
12 minutes ago  -  4 people like this.        PhotoBeitrag
  No comments.
```

dabei werden die Besonderheiten der Beitragsarten nicht berücksichtigt - der `photoBeitrag` hat keine Bilddatei und keine Caption. Eigentlich sollte das nämlich so aussehen:

```
Leonardo da Vinci
Had a great idea this morning.
But now I forgot what it was. Something to do with flying ...
40 seconds ago - 2 people like this.
  No comments.                                   TextBeitrag

Alexander Graham Bell
[experiment.jpg]                                 PhotoBeitrag
I think I might call this thing 'telephone'.
12 minutes ago  -  4 people like this.
  No comments.
```

Die spontane Lösungsidee verschiebt die `display`-Methode in die Subklassen, so dass jede Subklasse eine eigene `display`-Methode hat, welche dann natürlich entsprechend der spezifischen Eigenschaften implementiert sein könnte:



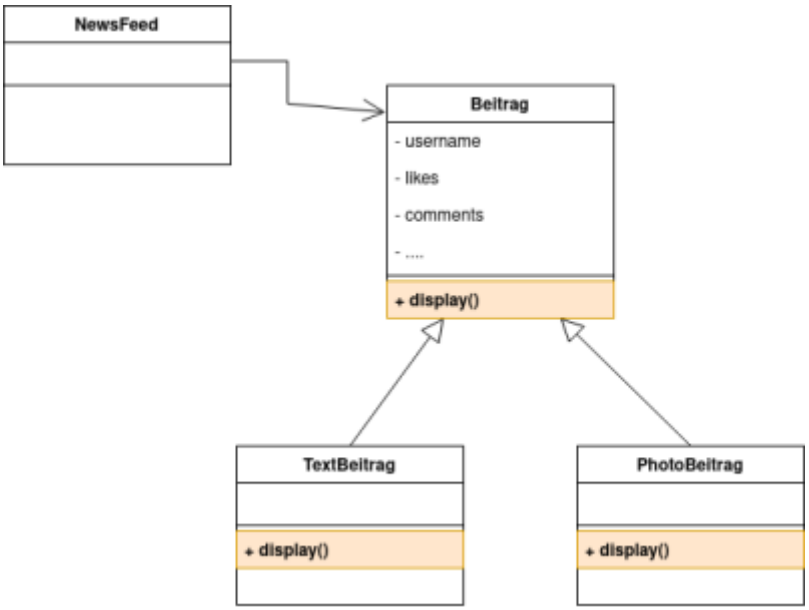
Dieser Versuch ist zum Scheitern verurteilt:

- Zugriff auf die privaten geeerbten Attribute aus Beitrag ist nicht möglich.
- Die Klasse `Newsfeed` benötigt eine `display`-Methode in Beitrag.

Lösungsansatz: Überschreiben

- Superklasse und Subklasse definieren Methoden mit gleicher Signatur.
- Jede der Methoden hat Zugriff auf alle Attribute (Felder) ihrer jeweiligen Klasse.
- Der Check des statischen Typs der Superklasse ist erfüllt.
- Die Methode der Subklasse wird erst zur Laufzeit aufgerufen und überschreibt dabei die Version der Superklasse.

Es gibt also `display`-Methoden in der Superklasse und in den Subklassen (wenn nötig):



Fragen:

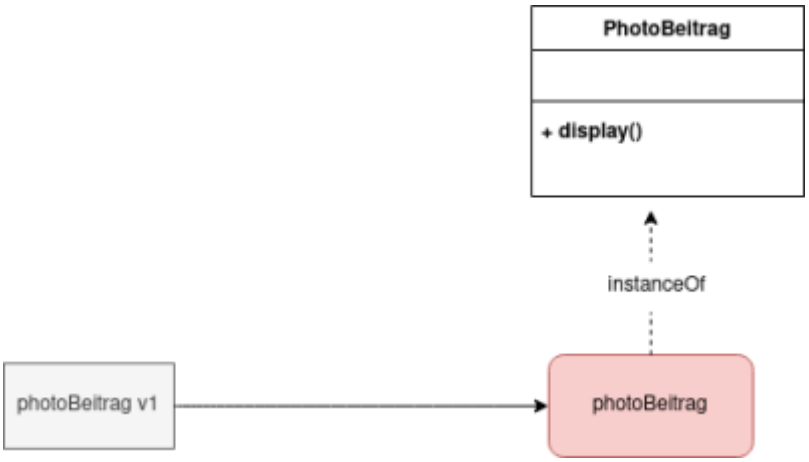
- Welche Rolle spielt die Version der Superklasse?
- Welche der `display`-Methoden wird denn zur Laufzeit tatsächlich aufgerufen?

Methodenauswahl zur Laufzeit

Ohne Vererbung

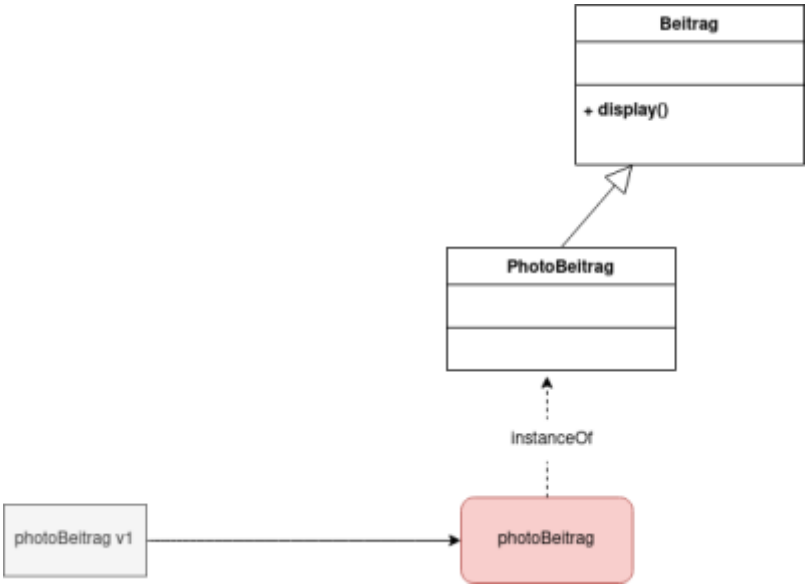
Keine Vererbung, kein Polymorphismus, kein Problem → die offensichtliche Methode wird ausgeführt:

```
v1.display()
```



Vererbung, kein Überschreiben

```
v1.display()
```

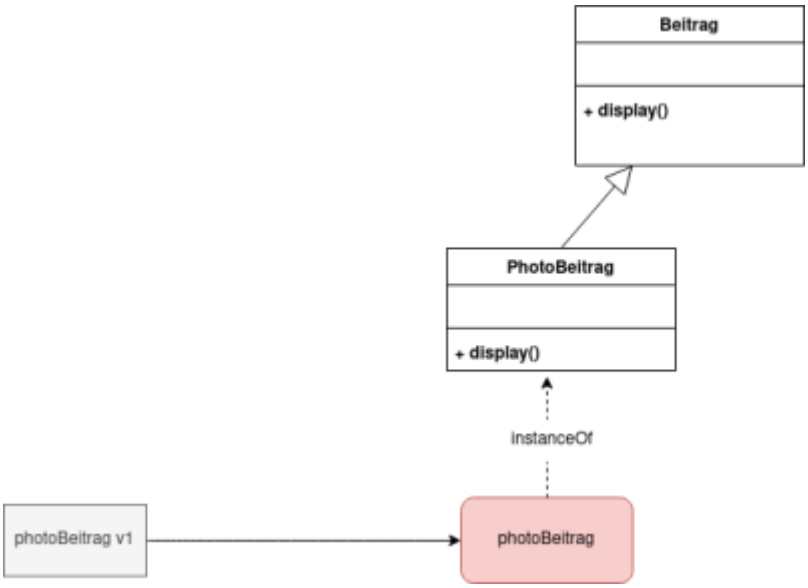


Bei der Suche nach der Methode wird die Vererbungshierarchie von unten nach oben durchlaufen (beginnend beim dynamischen Typ), bis zum Treffer – diese Methode wird ausgeführt.

> Die Methode der Superklasse wird also solange von den Versionen der Subklasse(n) „verdeckt“, solange es in den Subklassen Methoden mit gleicher Signatur gibt.

Vererbung und Überschreiben:

```
v1.display()
```



Bei der Suche nach der auszuführenden Methode wird die Vererbungshierarchie von unten nach oben durchlaufen (beginnend beim dynamischen Typ), **bis der erste Treffer gefunden wird** – diese Methode wird ausgeführt. Im Beispiel also die Methode, die in `photoBeitrag` implementiert ist.

Methodenwahl: Zusammenfassung

1. Auf die Variable wird zugegriffen
2. Das in der Variable gespeicherte Objekt wird aufgesucht
3. Die Klasse des Objekts wird bestimmt
4. Implementiert die Klasse die gesuchte Methode?
5. Wenn nicht, wird die Superklasse untersucht.
6. Schritte 4) und 5) werden solange wiederholt, bis die Methode gefunden wurde oder die Klassenhierarchie bis ganz nach oben durchlaufen wurde.
7. Die überschriebenen Methoden verdecken dabei stets die von der Superklasse geerbten Versionen.

Alle Klassen erben von "Object"

Jede Klasse erbt von Java Object, das kann man beispielsweise erkennen, wenn man einfach mal eine "leere" Klasse erstellt:

```
class Ball () {

}
```

Man kann diesen Ball instanziieren, obwohl er weder Konstruktor noch weitere Methoden enthält:

```
Hund ball01 = new Ball();
```

Er hat sogar schon Methoden, z.B. die Methode `toString()`:

```
ball01.toString();
```

das Erzeugt eine Ausgabe der folgenden Form:

```
ball01@af54f432
```

https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Object.html

Material

auswahl_102.png	26.4 KiB	29.11.2021 19:49
auswahl_103.png	41.1 KiB	29.11.2021 19:49
fahrzeuge.drawio.png	13.3 KiB	29.11.2021 14:55
kap11_polymorphie01.odp	1.5 MiB	29.11.2021 19:46
kap11_polymorphie01.pdf	360.2 KiB	29.11.2021 19:46
newssystem.drawio.png	13.1 KiB	29.11.2021 15:06
newssystem01.drawio.png	16.9 KiB	29.11.2021 19:52
newssystem02.drawio.png	18.2 KiB	29.11.2021 19:56
poly01.png	10.9 KiB	29.11.2021 19:59

poly02.png	14.6 KiB	29.11.2021 19:59
poly03.png	15.7 KiB	29.11.2021 19:59

From:
https://info-bw.de/ -

Permanent link:
https://info-bw.de/faecher:informatik:oberstufe:modellierung:vererbung:polymorphismus:start?rev=1638536489

Last update: **03.12.2021 13:01**