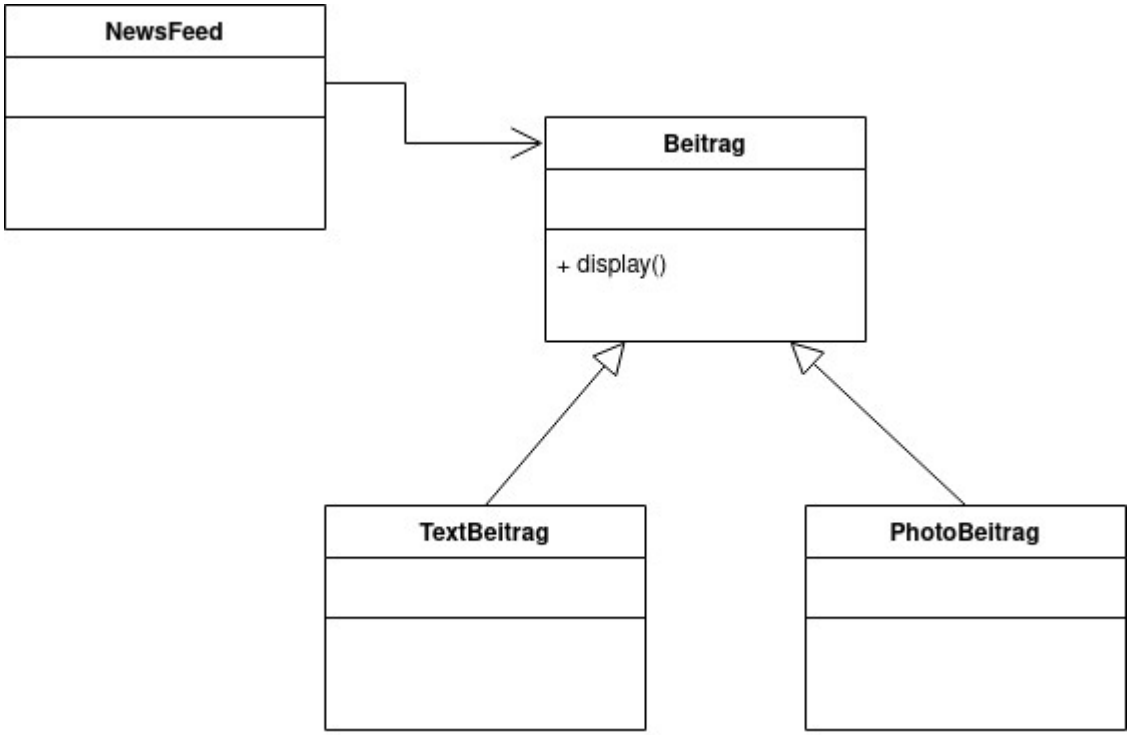
Vererbung und Polymorphie



Die Anzeigemethode `display` wird von `Beitrag` vererbt.

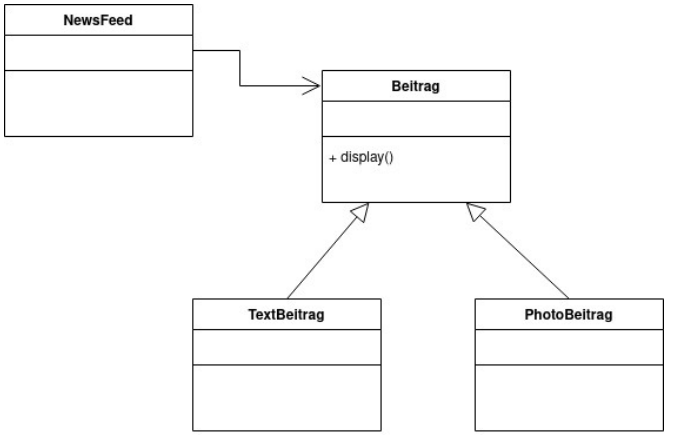
Die Anzeigemethode `display` wird von `Beitrag` vererbt – weiß nichts über besondere Eigenschaften der Subklassen:

```
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.
```

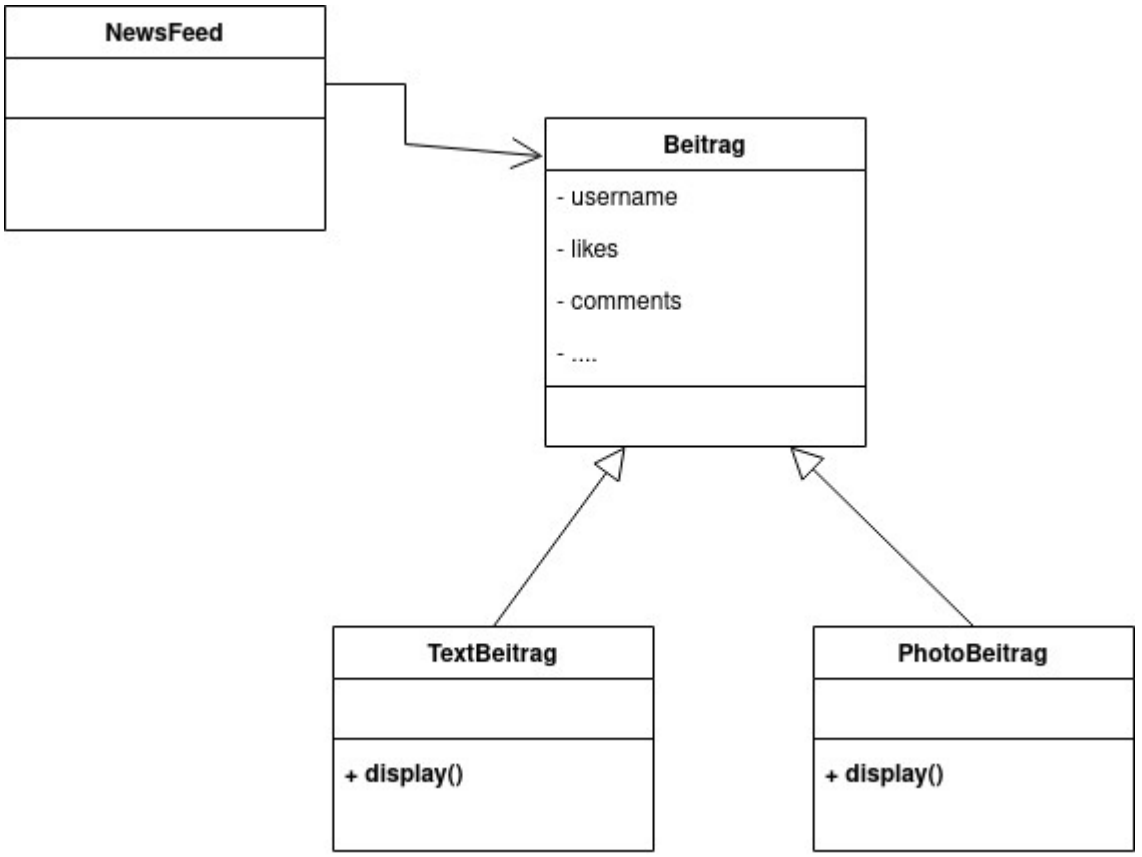
Die Ausgabe sollte die spezifischen Eigenschaften der Beitragsarten berücksichtigen und etwa so aussehen:

```
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 Superklasse weiß nichts über ihre Subklassen.
Vererbung ist eine Einbahnstrasse!



Eine vererbte Methode kann also prinzipiell besondere Eigenschaften von Unterklassen nicht berücksichtigen

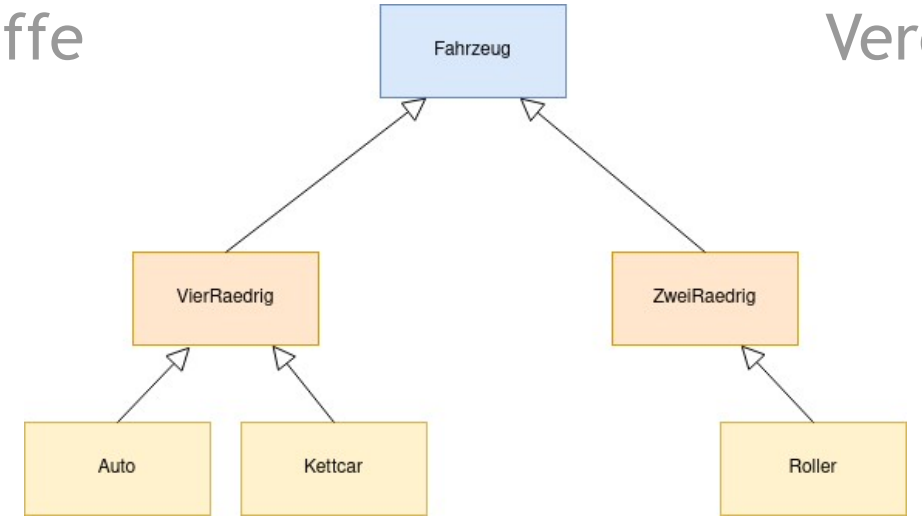


Eigene display-Methoden in den Subklassen:

Zugriff auf die privaten geeerbten Attribute aus `Beitrag`?

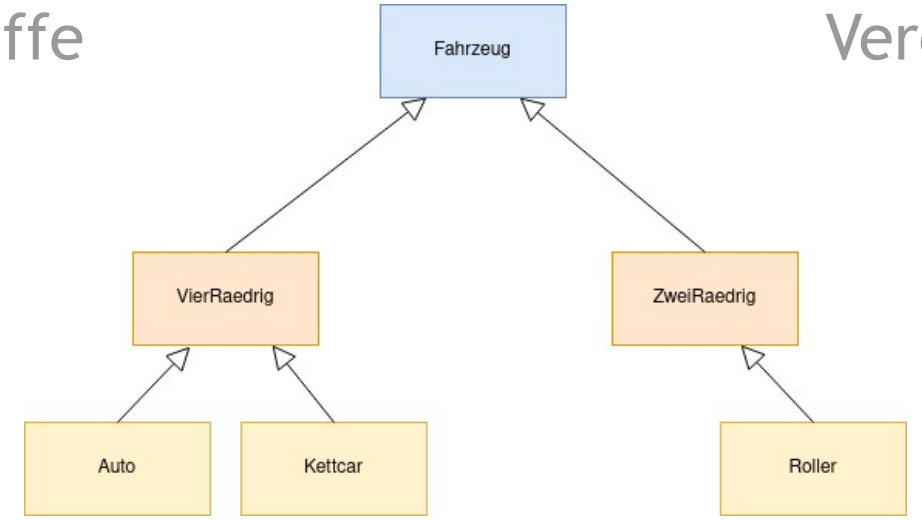
`Newsfeed` benötigt eine display-Methode in Beitrag?

So einfach gehts also nicht, man braucht noch weitere Konzepte um komplexere Vererbungshierarchien zu beschreiben.




```
Auto a1 = new Auto();    // Welchen Typ hat a1?
Fahrzeug f1 = new Auto();    // Welchen Typ hat f1?
```

Unterschied? Probleme?




```
Auto a1 = new Auto();    // Welchen Typ hat a1?

Fahrzeug f1 = new Auto();    // Welchen Typ hat f1?
```

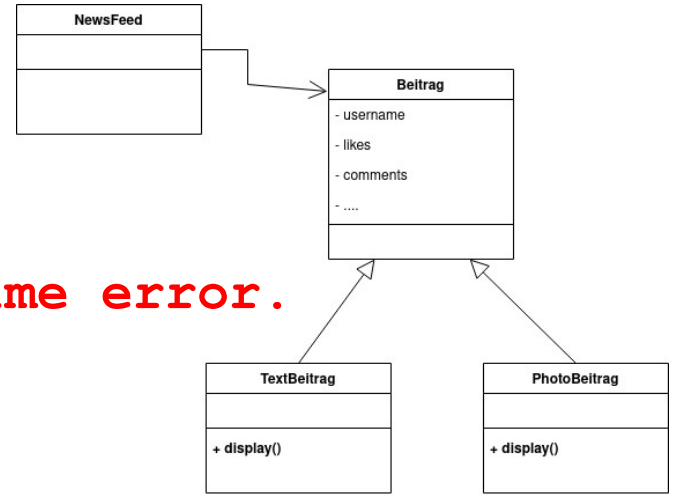
- Der **deklarierte** Typ einer Variablen ist ihr **statischer** Typ (static type).
- Der **Typ des Objekts, welches eine Variable hält** ist ihr **dynamischer** Typ (dynamic type)

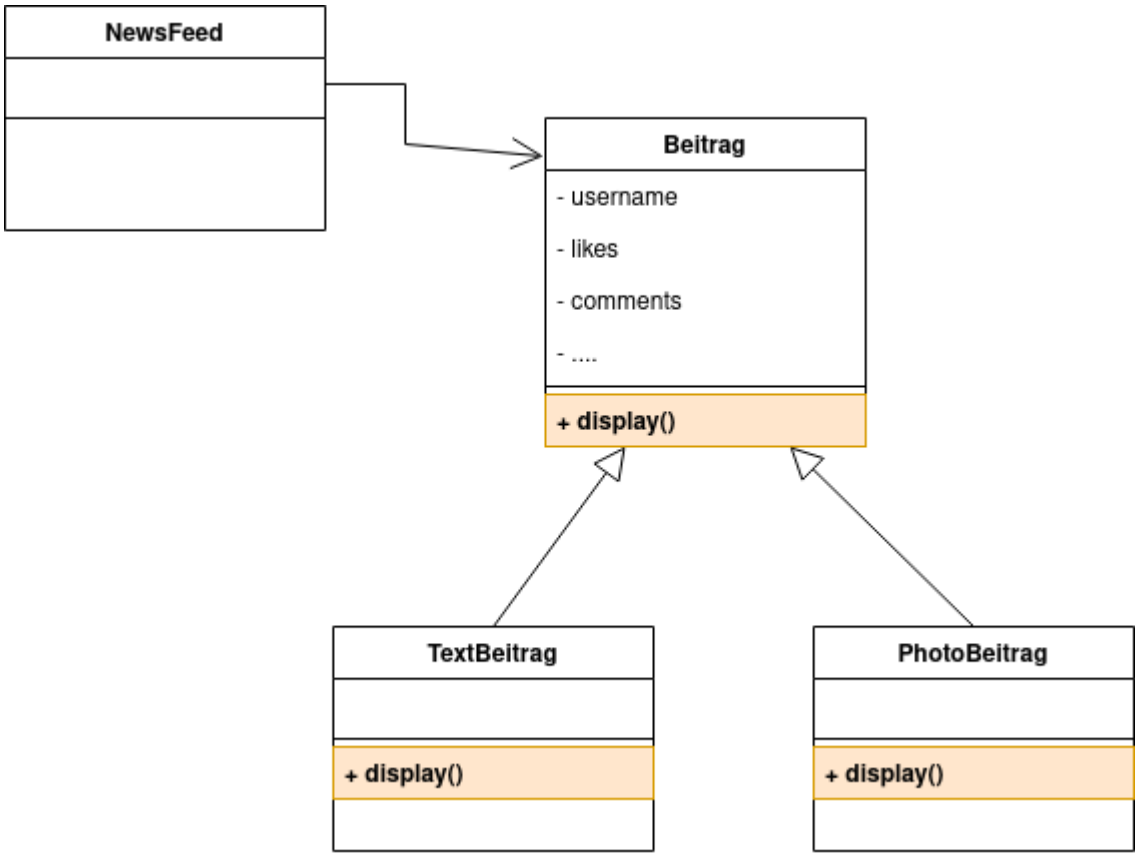
```
Auto a1 = new Auto();    // Welchen Typ hat a1?

Fahrzeug f1 = new Auto();    // Welchen Typ hat f1?
```

- Der **deklarierte** Typ einer Variablen ist ihr **statischer** Typ (static type).
- Der **Typ des Objekts, welches eine Variable hält** ist ihr **dynamischer** Typ (dynamic type)

Compiler findet Verletzungen bei statischen Typen.
Probleme bei dynamischen Typen gibt **Laufzeitfehler.**

```
for(Beitrag beitrag : posts) {
    beitrag.display();    // Compile-time error.
}
```



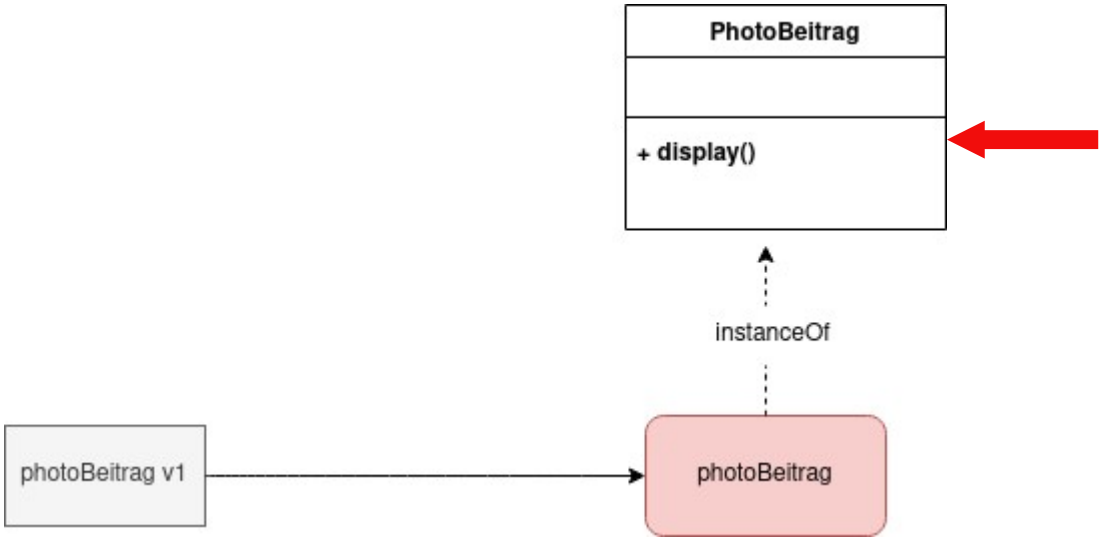


display-Methoden in der Superklasse und in den Subklassen:

Check des statischen und des dynamischen Typs funktioniert!

Dieses Vorgehen nennt man **Overriding** oder **Überschreiben.** Man **überschreibt** in der Subklasse eine Methode, die in der Superklasse bereits definiert ist.

- 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.
- Welche Rolle spielt die Version der Superklasse?

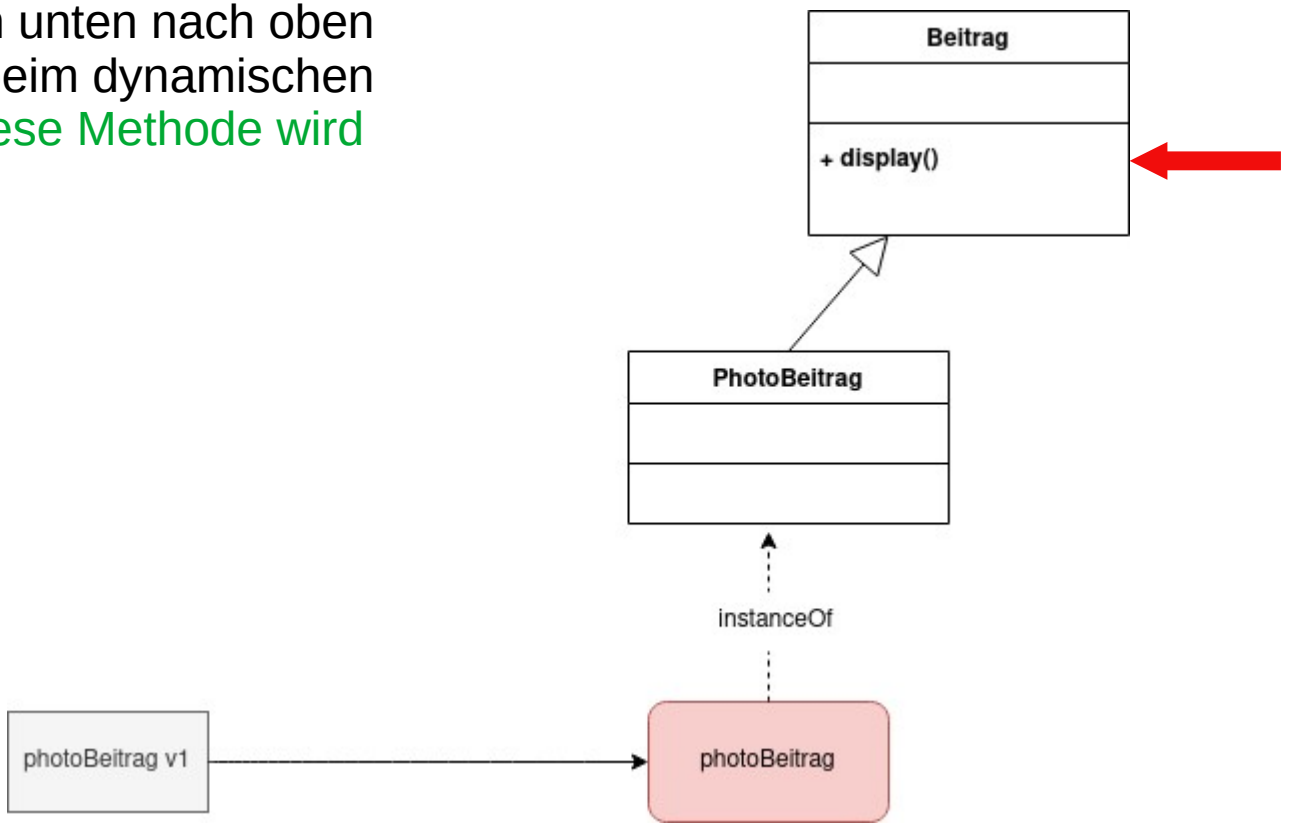



```
v1.display()
```

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

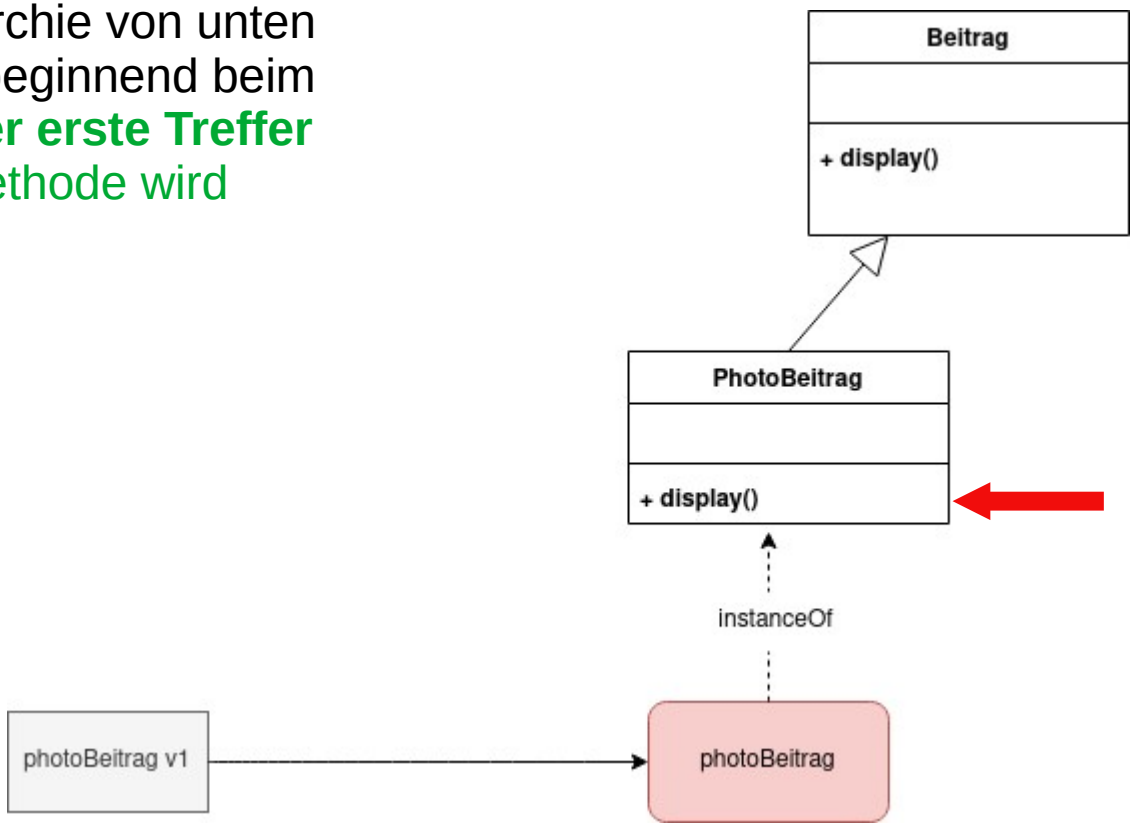
```
v1.display()
```

Vererbung, kein Überschreiben: 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.



```
v1.display()
```

Vererbung und Überschreiben: 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.



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

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.

Überschriebende Methoden werden verdeckt - manchmal möchte man aber explizit die Methodenversion der Superklasse aufrufen – was nun?

- Eine überschriebene Methode kann aus der überschreibenden Methode heraus aufgerufen werden.

```
public void methodenname(...) {

    super.methodenname(...);

    ...
}
```

- Vergleiche mit dem super-Aufruf bei den Konstruktoren!

```
// display of photoBeitrag
public void display()
{
    super.display();
    System.out.println(" [" + filename + "]");
    System.out.println(" " + caption);
}
```

Den Vorgang nennt man **polymorphe Methodenausführung**

- Eine polymorphe Variable kann Objekte verschiedener Typen speichern.
- Methodenaufrufe sind polymorph:

  Die Methode die zur Laufzeit aufgerufen wird hängt vom dynamischen Typ des Objekts ab.

Der **`instanceof`**-Operator wird verwendet um zur Laufzeit den dynamischen Typ einer Variablen zu ermitteln.

Kommt oft zum Einsatz, um Casts abzusichern:

```
if(post instanceof photoBeitrag) {
    photoBeitrag msg = (photoBeitrag) post;
    ... Zugriff auf die Methoden von photoBeitrag über msg ...
}
```

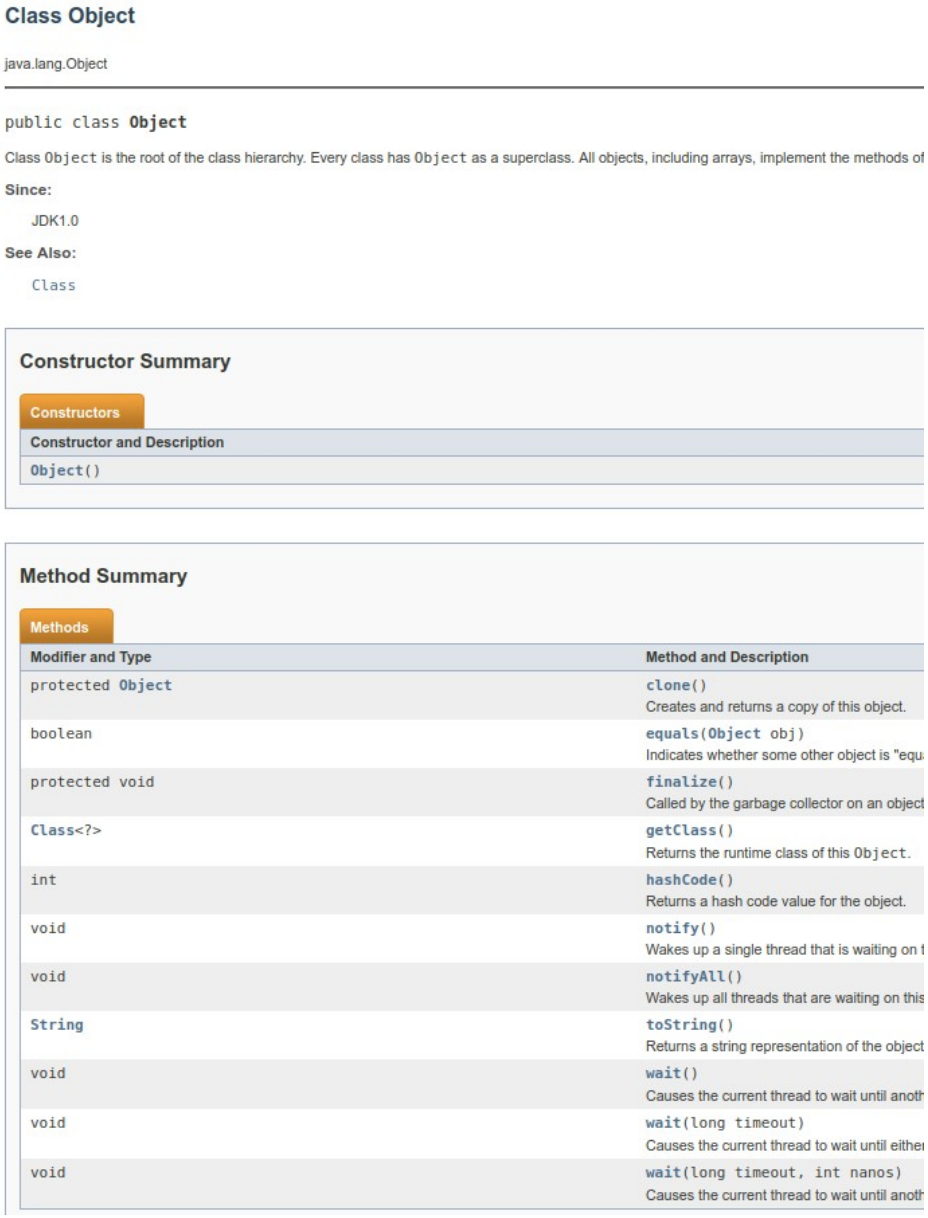
Anmerkung: Exceptions sollte man nicht für so was verwenden (Performancegründe).
→ Exceptionhandling für unplanbare Probleme!

Da alle Objekte von Java-Object erben, erben Sie auch die Methoden von Object → Auch diese Methoden können überschrieben werden.

Häufig überschrieben wird die **toString**-Methode:

```
public String toString()
```

Das wird dazu genutzt um eine String-Darstellung eines Objekts zu erhalten.



Class Object

java.lang.Object

`public class Object`

Class `Object` is the root of the class hierarchy. Every class has `Object` as a superclass. All objects, including arrays, implement the methods of

Since:

JDK1.0

See Also:

`Class`

Constructor Summary

Constructors

Constructor and Description
`Object()`

Method Summary

Methods

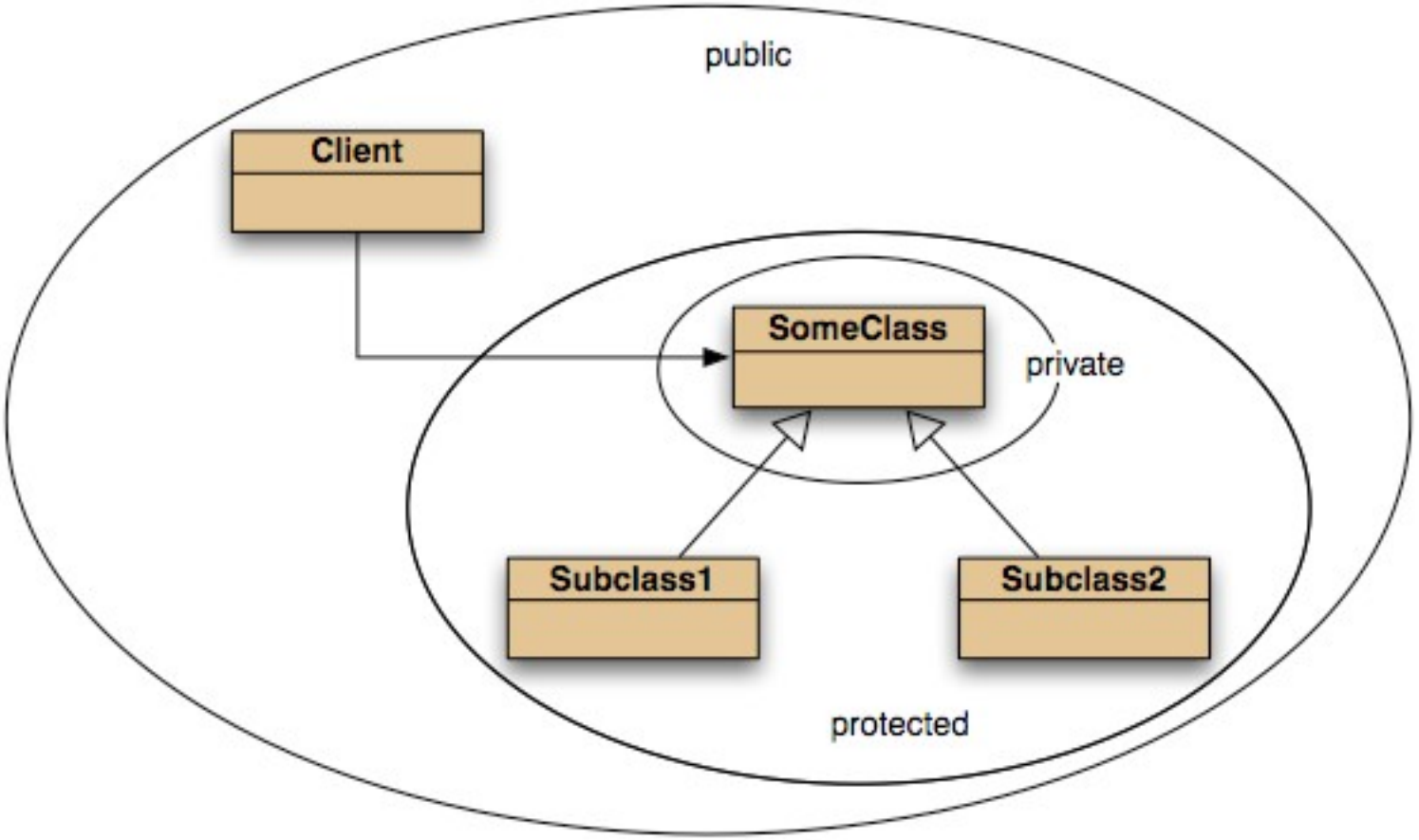
Modifier and Type	Method and Description
`protected Object`	`clone()` Creates and returns a copy of this object.
`boolean`	`equals(Object obj)` Indicates whether some other object is "equ
`protected void`	`finalize()` Called by the garbage collector on an object
`Class<?>`	`getClass()` Returns the runtime class of this `Object`.
`int`	`hashCode()` Returns a hash code value for the object.
`void`	`notify()` Wakes up a single thread that is waiting on t
`void`	`notifyAll()` Wakes up all threads that are waiting on this
`String`	`toString()` Returns a string representation of the object
`void`	`wait()` Causes the current thread to wait until anoth
`void`	`wait(long timeout)` Causes the current thread to wait until either
`void`	`wait(long timeout, int nanos)` Causes the current thread to wait until anoth

```
public String toString()
{
    String text = username + "\n" +
                  timeString(timestamp);
    if(likes > 0) {
        text += " - " + likes + " people like this.\n";
    }
    else {
        text += "\n";
    }
    if(comments.isEmpty()) {
        return text + " No comments.\n";
    }
    else {
        return text + " " + comments.size() +
               " comment(s). Click here to view.\n";
    }
}
```

Wenn man **`println`** mit einem Objekt als Argument aufruft, wird automatisch **`toString`** aufgerufen.

```
System.out.println(post);
System.out.println(post.toString());
```

- Manchmal sind private Attribute der Superklasse eine (zu) große Einschränkung
- Man kann den Subklassen Zugriff gewähren mit **geschütztem Zugriff** (protected access).
- `protected` ist eingeschränkter als `public` – nur Subklassen dürfen zugreifen.
- Best-Practice: Attribute `private` deklarieren, Zugriff mit Gettern und Settern die `protected` deklariert sind.
  ("Kapselung nach unten in der Vererbungshierarchie")


public
Client
SomeClass
private
Subclass1
Subclass2
protected