

Programmierkonzepte in der Physikalischen Chemie

25. Liskov-Substitutionsprinzip

Albert-Ludwigs-Universität Freiburg



**UNI
FREIBURG**

Dr. Till Biskup

Institut für Physikalische Chemie
Albert-Ludwigs-Universität Freiburg
Wintersemester 2018/19



- Objekte einer abgeleiteten Klasse sollten sich genauso verhalten wie Objekte der Basisklasse.
- Verletzungen des Liskov-Substitutionsprinzips lassen sich immer nur im konkreten Kontext feststellen.
- Das Liskov-Substitutionsprinzip ist eine wesentliche Voraussetzung für das Open-Closed-Prinzip.
- „Entwurf gemäß Vereinbarung“ (*design by contract*) formalisiert die Erwartungen an eine Klasse.
- Allgemein sorgt das Prinzip für klar definierte Schnittstellen und austauschbare Implementierungen auf allen Ebenen.

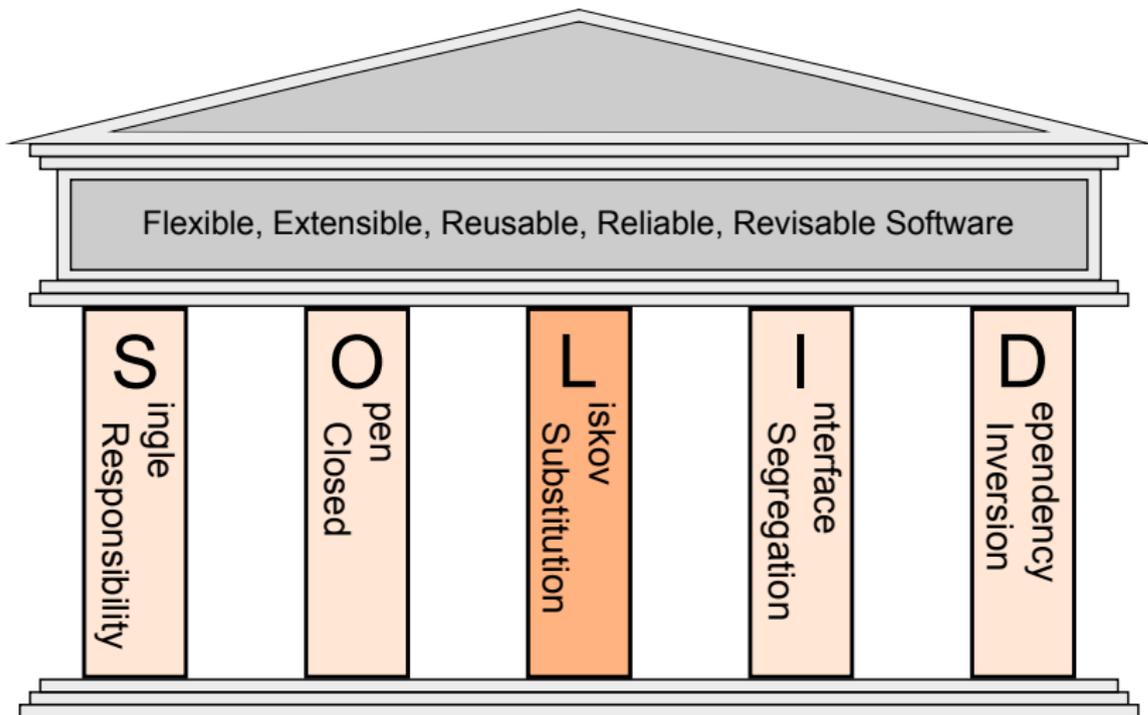
Das Liskov-Substitutionsprinzip

Beispiele für seinen Einsatz

Bedeutung im Gesamtkontext der Software-Architektur

Das Liskov-Substitutionsprinzip

Übersicht über die fünf Prinzipien



Die Originalformulierung

- “ *What is wanted here is something like the following substitution property [...]: If for each object o_1 of type S there is an object o_2 of type T such that for all programs P defined in terms of T , the behavior of P is unchanged when o_1 is substituted for o_2 , then S is a subtype of T .*
- Liskov, 1987

Eine etwas zugänglichere Formulierung

- “ *objects of the subtype ought to behave the same as those of the supertype as far as anyone or any program using supertype objects can tell.*
- Liskov und Wing, 1994

Liskov, *ACM Sigplan Not.* **23**:17–34, 1987

Liskov und Wing, *ACM Trans. Program. Lang. Syst.* **16**:1811–1841, 1994

- ▶ Typen sind im Kontext der OOP Klassen.
 - Subtypen sind abgeleitete Klassen.
 - Supertypen sind Klassen, von denen geerbt wird.
- ▶ Das erwartete Verhalten von Objekten steht im Fokus.
 - Ein Subtyp sollte sich genauso wie ein Grundtyp verhalten.
 - Ein Subtyp kann zusätzliches Verhalten aufweisen.
- ▶ Vererbung wird oft als *ist-ein*-Beziehung gesehen.
 - Das kann konzeptionell zu Problemen führen.
- 👉 Hinweise und Regeln für Vererbungshierarchien, die das Open-Closed-Prinzip ermöglichen

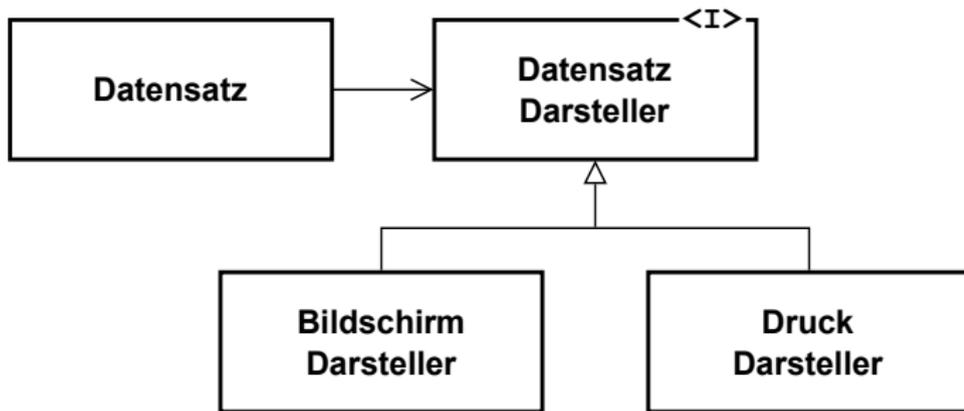
Open-Closed-Prinzip (OCP)

Eine Klasse sollte offen für Erweiterung,
aber verschlossen gegenüber Änderungen sein.

- ▶ Zusammenhang mit dem OCP
 - Abgeleitete Klassen implementieren abstrakte Methoden, die in (abstrakten) Basisklassen definiert wurden.
 - erfordert klare Regeln für die Vererbung
- ▶ Fragen, die adressiert werden:
 - Was muss man beim Einsatz von Vererbung beachten?
 - Welche Probleme können bei der Vererbung auftreten, die das Open-Closed-Prinzip verletzen?

Beispiele für seinen Einsatz

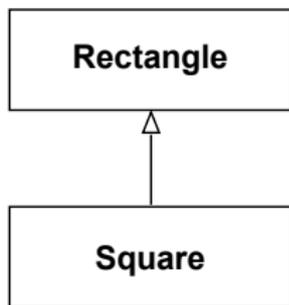
Eine Situation im Einklang mit dem LSP



→ Abhängigkeit

▷ Vererbung

<I> Schnittstelle



- ▶ Rectangle ist die Superklasse, Square die Subklasse.
 - Vererbung führt zu einer *ist-ein*-Beziehung.
 - *Geometrisch* ist ein Quadrat ein Rechteck.
- ▶ Entspricht die Relation dem Liskov-Substitutionsprinzip?
 - hängt vom Kontext und vom erwarteten Verhalten ab

Situation 1: Rechteck und Quadrat sind unveränderlich

- ▶ Höhe und Breite können nicht verändert werden.
 - nur Getter, aber keine Setter vorhanden
 - Höhe und Breite werden im Konstruktor gesetzt.
- ▶ Weiteres Verhalten ist nicht betroffen.
 - Bsp.: Fläche kann berechnet werden.
 - Darstellung etc. ist ebenfalls invariant.
- ☞ Square kann statt `Rectangle` verwendet werden, ohne dass anderer Code geändert werden müsste.
- ☞ Keine Verletzung des Liskov-Substitutionsprinzips.

Situation 2: Rechteck und Quadrat sind veränderlich

- ▶ Höhe und Breite können verändert werden.
 - Höhe und Breite sind im Rechteck unabhängige Größen.
 - Höhe und Breite sind im Quadrat immer identisch.
- ▶ Das Verhalten ist *unterschiedlich*.
 - Rechteck: Seiten sollten *unabhängig* änderbar sein.
 - Quadrat: Seiten sind immer gekoppelt – Änderung der Breite führt zwangsläufig zur Änderung der Höhe.
- ☞ Square kann *nicht* statt Rectangle verwendet werden, ohne dass es zu Überraschungen kommen kann.
- ☞ *Verletzung* des Liskov-Substitutionsprinzips.

Grundregel

Eine abgeleitete Klasse, die weniger tut als ihre Basisklasse, kann meist nicht anstelle der Basisklasse eingesetzt werden.

- ▶ explizite Typabfragen von Variablen etc.
- ▶ degenerierte Methoden in abgeleiteten Klassen
 - Grund: Methode in der abgeleiteten Klasse nutzlos
 - führt nicht immer, aber oft zu Problemen
 - Symptom für Verletzung des Interface-Segregation-Prinzips
- ▶ neue Ausnahmen (*exceptions*) in abgeleiteten Klassen
 - Vorbedingungen dürfen für abgeleitete Klassen nicht strikter sein als für die Basisklasse.



- ▶ legt Kriterien für Vererbungsbeziehungen fest
 - Austauschbarkeit abgeleiteter Klassen mit der Basisklasse
 - gleiches Verhalten aus Sicht der Basisklasse
- ▶ ermöglicht das Open-Closed-Prinzip
 - beruht auf Vererbung von abstrakten Klassen
 - Die abstrakte Basisklasse definiert das Verhalten, an das sich die abgeleiteten Klassen halten müssen.
 - führt zu Flexibilität, Wiederverwendbarkeit, Wartbarkeit
- ▶ „Entwurf gemäß Vereinbarung“ (*design by contract*)
 - formalisiert die Erwartungen an eine Klasse und die von ihr abgeleiteten Klassen
 - lässt sich in den wenigsten Sprachen direkt formulieren

- ▶ zwei Arten von Erwartungen
 - Vorbedingung: Voraussetzung für einen Methodenaufruf
 - Nachbedingung: garantierter Zustand nach dem Aufruf

- ▶ Anforderungen an abgeleitete Klassen
 - dürfen keine strengeren Vorbedingungen fordern
 - müssen mindestens die von der Basisklasse vorgegebenen Nachbedingungen erfüllen

- ▶ Formulierung von Erwartungen
 - in den wenigsten Sprachen direkt möglich
 - Unittests sind eine gute Möglichkeit
 - ggf. in der Basisklasse explizit dokumentieren



- ▶ Fokus: Schnittstellen und ihre Implementierungen
 - ursprünglich als Regel für Vererbung in der OOP gesehen
 - Schnittstellen auch zwischen Komponenten, Schichten, ...
- ▶ Schnittstellen können sehr unterschiedlich aussehen.
 - Schnittstelle, die von mehreren Klassen implementiert wird
 - Klassen, die die gleichen Methodensignaturen teilen
 - Services, die alle der gleichen Schnittstelle antworten
- ▶ Gründe für die Anwendbarkeit des LSP
 - Nutzer hängen von wohldefinierten Schnittstellen ab.
 - Die Implementierungen dieser Schnittstellen sollten untereinander austauschbar sein.
- ☞ Eine einfache Verletzung der Austauschbarkeit kann viele zusätzliche Mechanismen notwendig machen.



- Objekte einer abgeleiteten Klasse sollten sich genauso verhalten wie Objekte der Basisklasse.
- Verletzungen des Liskov-Substitutionsprinzips lassen sich immer nur im konkreten Kontext feststellen.
- Das Liskov-Substitutionsprinzip ist eine wesentliche Voraussetzung für das Open-Closed-Prinzip.
- „Entwurf gemäß Vereinbarung“ (*design by contract*) formalisiert die Erwartungen an eine Klasse.
- Allgemein sorgt das Prinzip für klar definierte Schnittstellen und austauschbare Implementierungen auf allen Ebenen.