

Software Engineering 1



Anna Prenzel

**Ich wünsche Ihnen ein glückliches,
gesundes und erfolgreiches neues
Jahr!**

3 Grundsätze

- ↻ fürs wissenschaftliche Arbeiten
- ↻ aus eigener Erfahrung gewonnen

Grundsatz 1

Zeitpläne sind schädlich.

☞ „Ich überlege den ganzen Tag, wie ich die erste halbe Seite von Abschnitt XY schreiben soll. ~~Das ist schlecht, denn laut Zeitplan hätte ich heute 5 Seiten schreiben müssen.~~“

Grundsatz 2

Abstinenz bringt nichts.

☞ ~~„Nächste Woche schließe ich mich ein und schreibe den ganzen Beleg fertig. Diese Woche mache ich noch nichts.“~~

☞ ~~„Heute ist schönes Wetter, aber ich verzichte auf die Radtour, weil ich gerade mit dem Beleg nicht vorwärts komme.“~~

Grundsatz 3

**Korrigiere deine eigene Arbeit so,
als wäre sie die deines größten
Konkurrenten.**

∞ Ich könnte etwas verbessern, ~~aber ich~~
~~schreibe den Unterabschnitt nicht nochmal~~
~~neu, weil ich eine Woche dran gearbeitet habe~~
~~und so viel Mühe damit hatte.~~

Gute Vorsätze

Evaluation

Auswertung → Website

Wiederholung

∞ Liskov Substitution Prinzip

- Kovarianz/Kontravarianz
- Kreis/Ellipse-Problem

∞ Dependency Inversion

∞ „Böse ifs“

- Überflüssige Überprüfung von Vorbedingungen
- Verletzung des Open-Closed-Prinzips (stattdessen Polymorphie anwenden)

Wiederholung – Beispiel 1

```
public class A {}
```

```
public class SubA extends A {}
```

```
public class SubSubA extends SubA {}
```

Wiederholung – Beispiel 1

```
public class Test1{
```

```
    public void run(SubA sa){ ... }
```

```
}
```

```
public class SubTest1 extends Test1{
```

```
    public void run(SubSubA ssa){...}
```

```
}
```

Das ist Kovarianz von Parametern, d.h. die Vererbungsrichtung von SubSubA und Sub A entspricht der Vererbungsrichtung von SubTest1 und Test1.

Dies ist in Java nicht zulässig. Nur SubA ist zulässig. Grund: nächste Folie

Wiederholung – Beispiel 1

```
public static void main(String[] args){
```

```
    Test1 t1 = new Test1();
```

```
    SubA sa = new SubA();
```

```
    t1.run(sa);
```

```
}
```

Angenommen, wir möchten Test1 durch SubTest1 austauschen. Dann muss weiterer Code geändert werden: der Typ von sa muss zu SubSubA geändert werden.

Also kann Test1 nicht durch seinen SubTyp ersetzt werden → Verletzung des Liskov Substitution Prinzip.

Wiederholung – Beispiel 1

```
public class Test1{
```

```
    public void run(SubA sa){ ... }
```

```
}
```

```
public class SubTest1 extends Test1{
```

```
    public void run(A ssa){...}
```

```
}
```

Das ist Kontravarianz
(entgegen der
Vererbungsrichtung).

Das würde theoretisch
funktionieren: In der
main-Methode könnte
Test1 durch SubTest1
ohne weitere
Änderungen
ausgetauscht werden.

Ist in Java aber trotzdem
nicht erlaubt.

Wiederholung – Beispiel 2

```
package pack1;
```

```
public class Services {
```

```
    public void service1(){...}
```

```
    public void service2(){...}
```

```
    public void service3(){...}
```

```
}
```

Wiederholung – Beispiel 2

```
package pack2;  
import pack1.Services;  
  
public class Service1User {  
  
    public Services services;  
  
    public void run(){  
        this.services.service1();  
    }  
}
```

Angenommen,
Service1User interessiert
sich nur für die Funktion
service1(). Alle anderen
Funktionen sind nicht
relevant.

Hier ist **Service1User**
davon abhängig, dass
die Klasse **Services** im
fremden Paket existiert
und die Funktion
bereitstellt. Wenn die
Klasse umbenannt oder
in neue Klassen zerlegt
wird, muss hier der Code
geändert werden.

Wiederholung – Beispiel 2

```
package pack2;  
import pack1.Services;  
  
public class Service1User {  
  
    public Service1Provider services;  
  
    public void run(){  
        this.services.service1();  
    }  
}
```

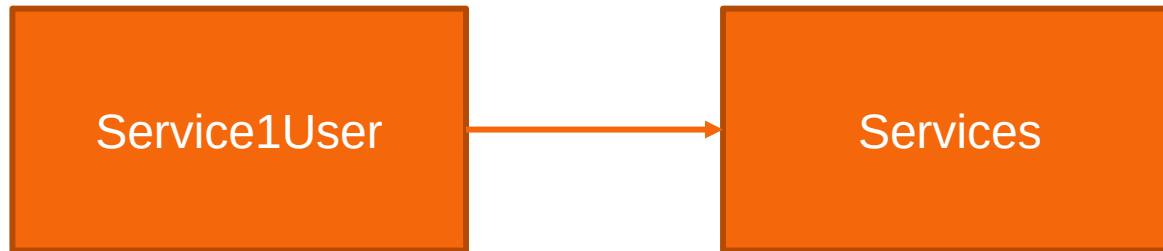
Besser: **Service1User** legt (z.B. in pack2) eine Schnittstelle fest, in der nur die Funktion **service1** enthalten ist.

Im anderen Paket muss diese Schnittstelle implementiert werden → egal von welcher Klasse. Änderungen im Code von pack1 werden in der Klasse **Service1User** nicht mehr bemerkt.

Das ist **Dependency Inversion** (siehe nächste Folie)

Wiederholung – Beispiel 2

Vorher



Nachher

