Delegation

Unter Delegation versteht man, wenn ein Klasse eine gewisse Methode nicht selbst implementiert, sondern nur dieselbe Methode von einer anderen Klasse aufruft, auf welche sie eine Referenz hat. Diese Referenz auf das andere Objekt der anderen Klasse ist als Member Variable in Klasse definiert.

Schaut euch dazu das "Example 2" auf der Seite https://www.javaguides.net/2018/08/delegation-in-java-with-example.html an. Dort wird das Problem angegangen, dass man mehrere Printertypen ansprechen möchte mit einem "generischen" PrintController.

Dieser PrintController soll nun ganz allgemein Anpassungen des Ausdrucks machen können (z.B. Booklet Printing, ...) durch einen PrinterFilter. Dieser Filter soll für alle Printertypen verfügbar sein. Durch das, dass der PrinterController das Interface Printer implementiert, kann er genauso angesprochen werden wie die "spezifischen" Printer Klassen CanonPrinter, EpsonPrinter und HpPrinter. Aber wir könnten jetzt auch zusätzlich die Funktionalität des "Booklet Layout" in einer Klasse BookletLayoutFilter des PrinterFilter Interfaces zum PrinterController dazufügen:

```
public class PrinterController implements Printer,PrinterFilter {

     private Printer printer;
     private PrinterFilter[] filters=null;

     public PrinterController(Printer printer) {
          this.printer = printer;
     }
     @Override
     public void print(String message) {
          printer.print(filter(message)); //Delegation an den Printer
     }
     private String filter(String message) {
     for(PrintFilter f: filters)
         {
               message=f.filter(message); //Delegation an den Filter
         }
              return message;
     }
     public void setFilters(PrinterFilter[] filters){
         this.filters=filters;
     }
}

public interface PrinterFilter {
     String filter(String message);
}
public class BookletFilter {
       public String filter(String message) {
            /* Hier müsste die Änderungen am Dokument gemacht werden damit es
               ein Booklet Dokument ist statt ein gewöhnliches Dokument */

            return messagemodified;
       }
}
```

Der PrinterController kann nun Eigenschaften von 2 Interfaces erben und diese können erst noch wie plugins erweitert werden. D.h. ich kann ein neuen Printer Typ dazufügen, kann aber auch einen neuen Filter dazufügen, der etwas anderes mit dem Layout macht, bzw. mit dem Text.
Wir haben so ein Klasse PrinterController erzeugt, die sich so verhält wie ein physischer Printer, aber erweitert wurde durch die Filtermöglichkeit.

Das Dazufügen von neuen Eigenschaften zum PrinterController ist einfach und flexibel. Es ist nicht an eine Vererbungs-Hierarchie gebunden. Der einzige Nachteil in Java sind ein paar Zeilen mehr Code.