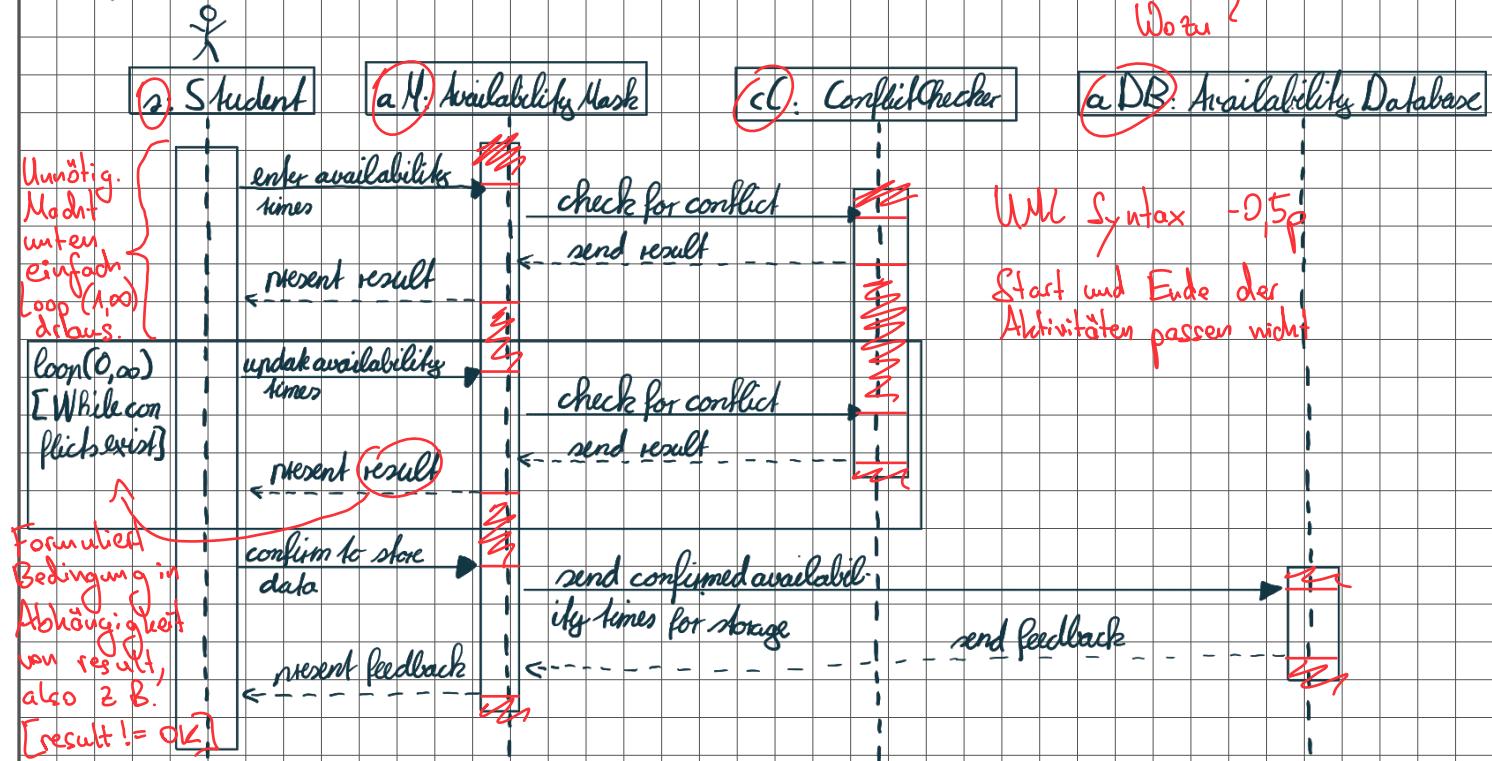


Blatt 4 - Bearbeitung

Nr 1



Nr 2 a)

context Course Manager:: addCourse(courseName: String) pre: courseName <=> null and !self.courses.contains(courseName) ✓

context Course Manager:: addCourse(courseName: String) post: self.courses.contains(courseName) and self.courses.size() = self.courses@pre.size() + 1 ✓

context Course Manager inv: !self.courses.contains(null) and self.courses.isUnique(c/c)

b)

Single Responsibility Principle: Das Prinzip ist erfüllt, da die einzige Funktion der Klasse Course Manager das verwalten (also hinzufügen, löschen etc.) von Kursen ist.

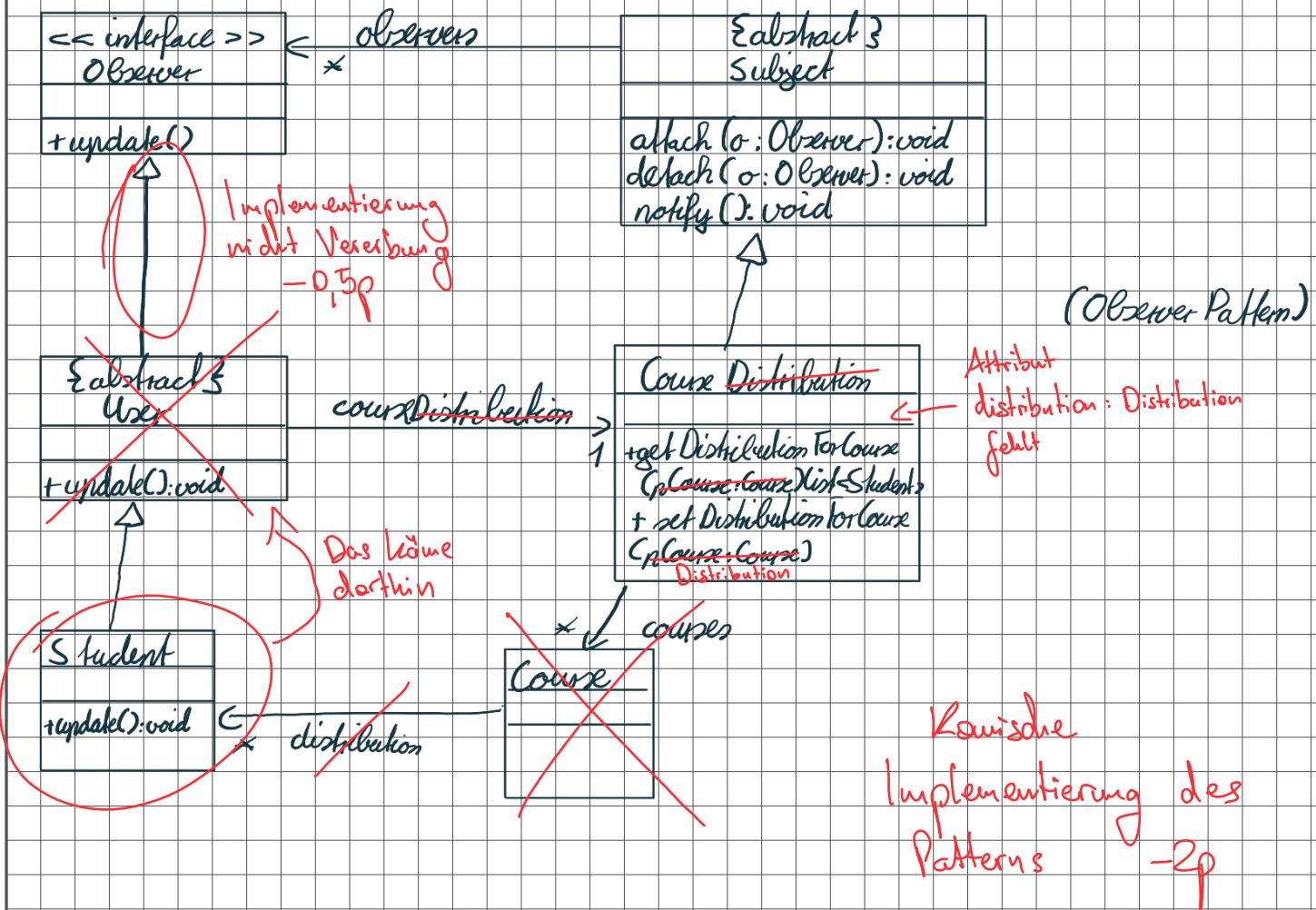
Open/Closed Principle: Da die Methoden addCourse und removeCourse final sind, sind diese nicht offen für Erweiterungen, wenn andere Klassen von Course Manager erben. Als Verbesserung könnte man die Methode nicht final machen, damit erende Klassen die Methode überschreiben und erweitern können.

Liskov Substitution Principle: Da keine Vererbung im Code vorliegt ist das Prinzip erfüllt.

Interface Segregation Principle: Der Grundsatz ist nicht anwendbar, da nicht bekannt ist, welche Klassen auf welche Teile vom Course Manager zugreifen. Könnten bzw. gewisse Klassen nur Kurse hinzufügen, wäre es sinnvoll die Klasse Course Manager über Interfaces aufzuteilen.

Dependency Inversion Principle: Gegen das Prinzip wird verstößen, da die Kursliste direkt mit der Unterklasse ArrayList und nicht mit dem Interface List initialisiert wird. ✓

Nr 3



Nr. 4



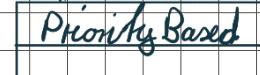
paradigm interface oder abstract class -0,5 p



```
+ distribute(course: Course)
```



```
+ distribute(course: Course)
```



```
+ distribute(course: Course)
```

(Strategy Pattern)