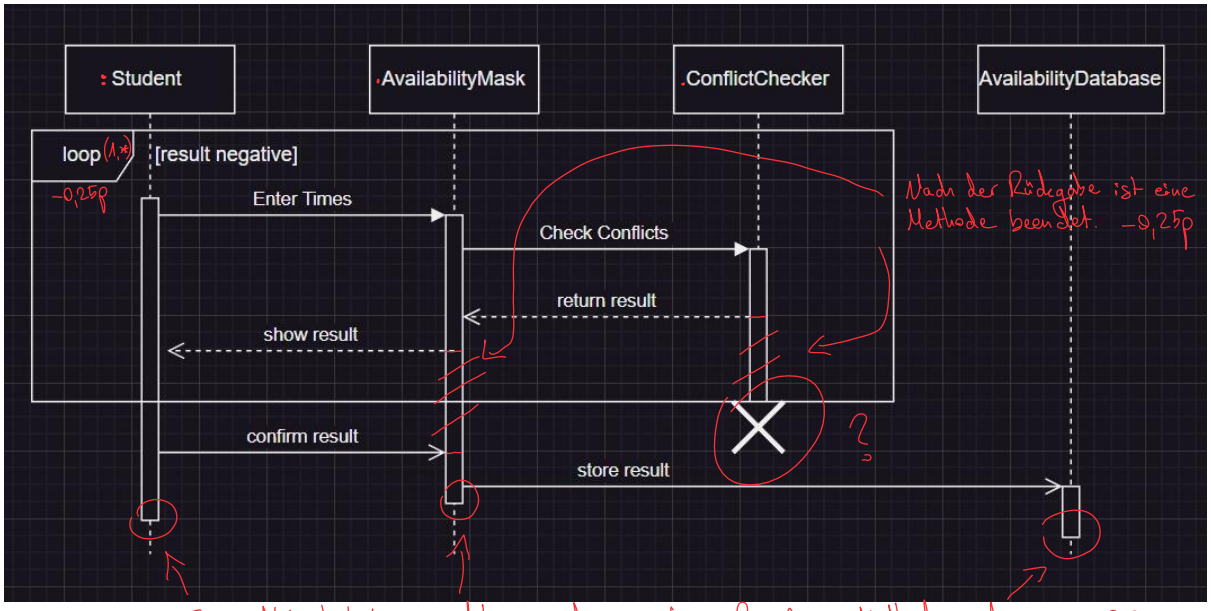
Homework 4

Aufgabe 1:



Aufgabe 2:

Invariant of the class:

context CourseManager inv: courseName != null

-- Alle Kursnamen in der Liste dürfen nicht null sein

context CourseManager inv: isUnique(courseName)

-- Die Liste courses darf keine Duplikate enthalten

Das müsste dann jeweils im OCL Statement auch auftauchen. -0,25p

Pre-and Postconditions of the Method addCourse():

context CourseManager::addCourse(courseName : String) pre: courseName != null

context CourseManager::addCourse(courseName : String)pre: courses !contains(courseName)

Kann mit "and" dazwischen zusammengefasst werden.

-- Der Kursname darf nicht null sein und nicht bereits in der Liste enthalten sein

context CourseManager::addCourse(courseName : String)post: courses contains(courseName)

-- Nach der Ausführung der Methode muss der Kursname in der Liste enthalten sein

✓

Pre-and Postconditions of the Method removeCourse():

context CourseManager::removeCourse(courseName : String) pre: courseName != null

context CourseManager::removeCourse(courseName : String)pre: courses contains(courseName)

-- Der Kursname darf nicht null sein und muss bereits in der Liste enthalten sein

context CourseManager::removeCourse(courseName : String)post: courses !contains(courseName)

-- Nach der Ausführung der Methode muss der Kursname in der Liste entfernt worden sein

b)

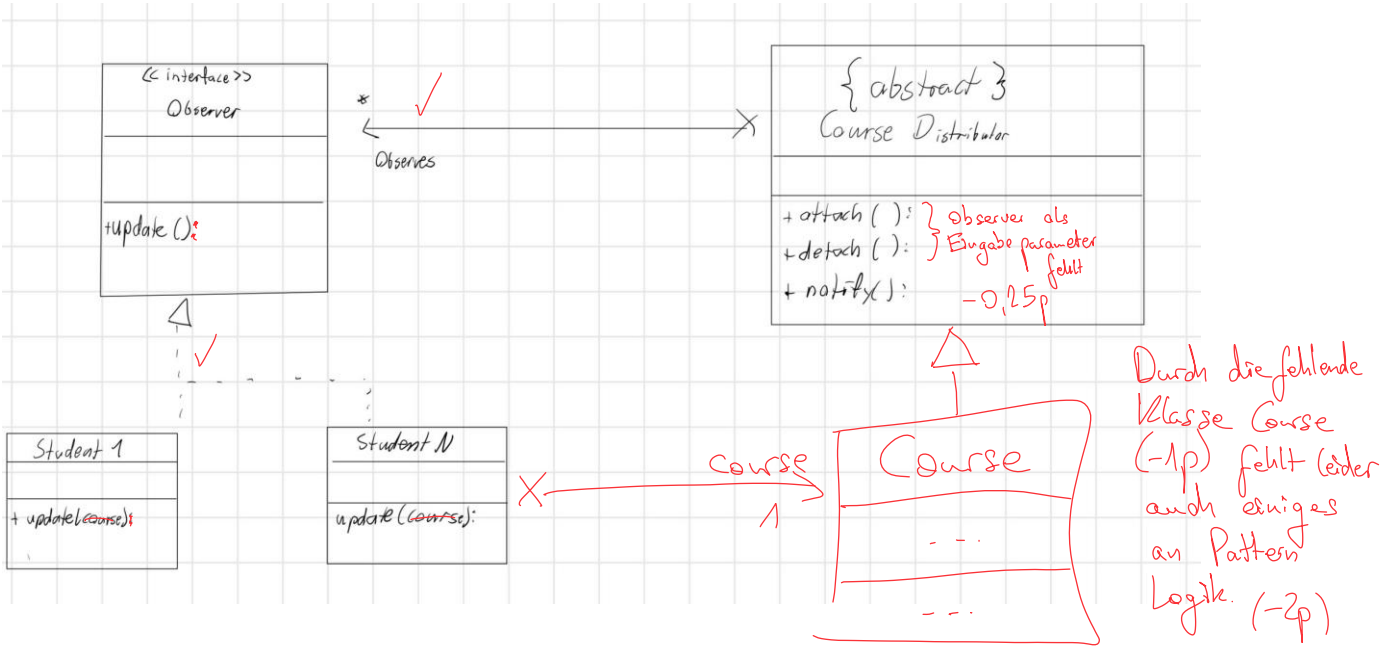
1. Single Responsibility Principle (SRP)
   - Ja die Klasse ist bisher nur für das Hinzufügen und Löschen von Klassen verantwortlich – eine Sache/Aufgabenteil. Dies kann sich ändern, wenn Methoden usw. hinzugefügt werden. Kein Verstoß.
2. Open-Closed Principle (OCP)
   - Da die Methoden addCourse() und removeCourse() in der Klasse deklariert sind, kann es sein, dass wenn die Bedingungen beim Hinzufügen/Löschen geändert werden, die Methode auch umgeschrieben werden müssen, was zum Verstoß des OCP führt. Dafür kann man Interfaces verwenden, wo die Methoden deklariert werden. Also bisher nicht sehr flexibel für Erweiterungen. (Verstoß)
3. Liskov Substitution Principle (LSP)
   - Kontext über Sub-und Superklassen fehlt uns bzw. keine Vererbung wird verwendet. (Bisher kein Verstoß)
4. Interface Segregation Principle (ISP)
   - Wir verwenden keine Interfaces, also auch da kein Verstoß gegen das Prinzip.
5. Dependency Inversion Principle (DIP)
   - Die Klasse ist von Arraylist<> abhängig (Z.6), was ein Verstoß ist, da es eine Implementierung der Schnittstelle List ist. Deswegen dürfen wir von ArrayList nicht abhängig sein und stattdessen List verwenden. (Verstoß)

Problematisch ist an der Stelle vor allem, dass die Methoden "final" sind.

✓

Aufgabe 3 (Observer Pattern):

Das Observer-Pattern dient dazu, Daten an verschiedenen Stellen konsistent zu halten, Events zu erzeugen und Klassen voneinander zu entkoppeln. In unserer Aufgabe ist dieses Muster besonders gut geeignet, da die Klasse CourseDistributor entkoppelt werden soll.



Aufgabe 4 (Strategy Pattern):

Wir haben uns für das Strategy-Pattern entschieden, da es sich hervorragend eignet, um Funktionen flexibel austauschbar zu gestalten und bei Bedarf neue hinzuzufügen, wie es in der Aufgabenstellung gewünscht ist.