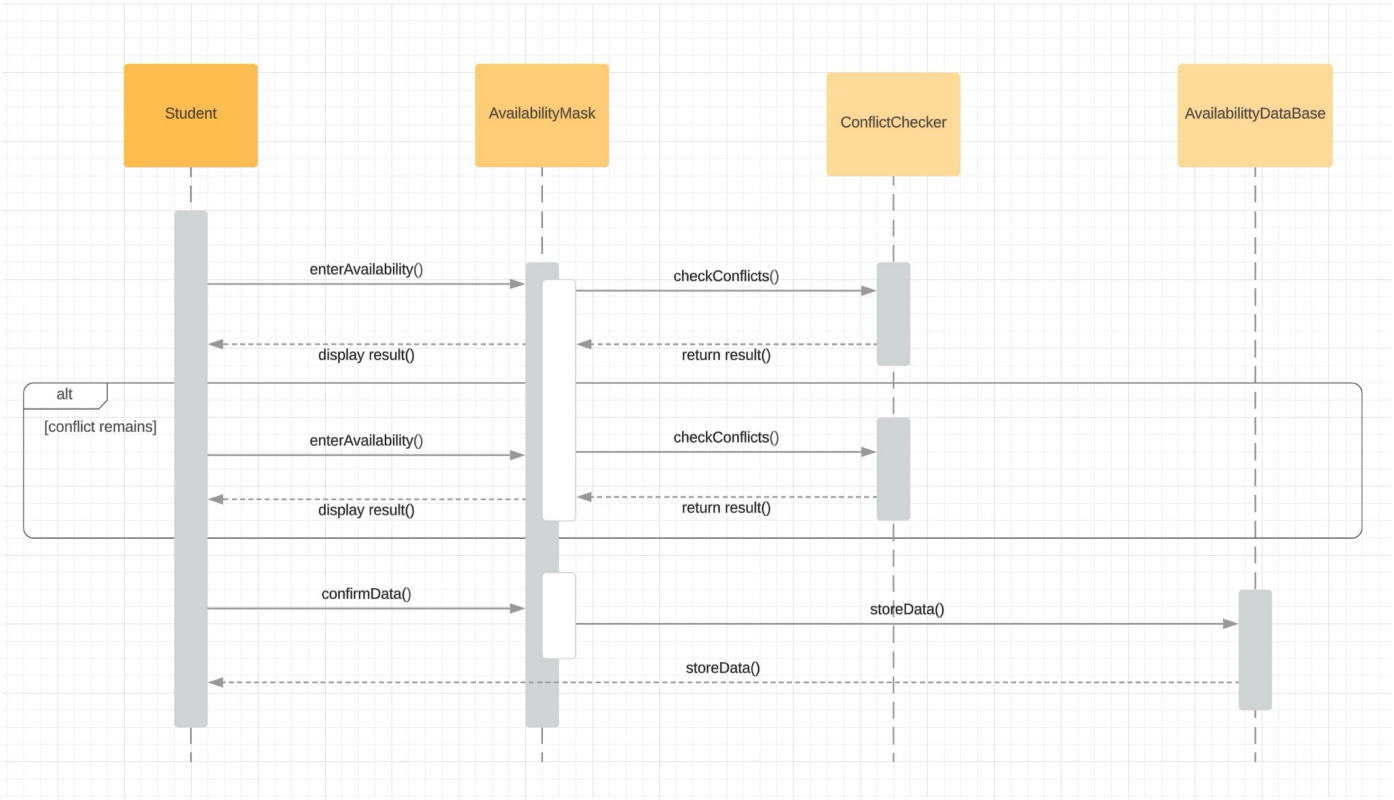
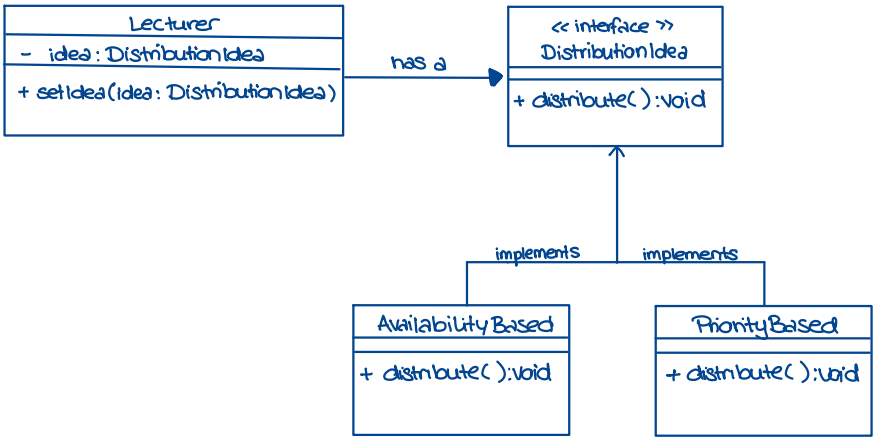
Homework 4

Amal Bolakhrif (7423946), Beyza Karaca (7422370), Lena Igoumrane (7424588)

Task 1



task 4



Amal Bolakhrif (7423946), Beyza Karaca (7422370), Lena Igoumrane (7424588)

Homework 4 Exercise 2

a)
siehe Klasse in *homework04/src/main/java/hw04/ex02a*

b)

Single Responsibility Principle (SRP)

- CourseManager class has a single responsibility: managing a list of courses (maintains internal course list, has methods to add/ remove courses)
- Checking of invariants or pre-/ postconditions not included in the class or the specific methods (for example checking if a course is null or already in list before adding it)
  - → Violates principle bc leaves responsibility of ensuring data integrity to the user of the class/ caller of the specific methods rather than enforcing it itself
  - Possible solution: modify methods with if-condition and/ or throwing an exception

```
public final void addCourse(String courseName) {
        if (courseName == null || courses.contains(courseName)) {
            return; // or throw an exception
        }
        courses.add(courseName);
}
```

Open/Closed Principle (OCP)

- Class not designed in a way that makes it easy to e.g. extend functionality without modifying lots of existing code when requirements
- e.g. if we wanted to change what the courses list stores (e.g., not Strings/ coursenames), we'd need to change the existing code directly.
- Possible solution: Generics

```
public class CourseManager<T> {
      private final List<T> courses;

      public CourseManager() {
            this.courses = new ArrayList<>();
      }

      public void addCourse(T course) {
            if (course == null || courses.contains(course)) {
                  throw new IllegalArgumentException("Item is invalid
                  or already exists!");
            }
            courses.add(course);
      }
      …
}
```

Liskov Substitution Principle (LSP)

- Code doesn't include inheritance → principle not really applicable.
- Anyways, use of keyword `final` in methods `addCourse` and `removeCourse` stops potential subclasses from overriding/ modifying them

- → no direct violation of LSP in code, but design might be too restrictive for extensions.

Interface Segregation Principle (ISP)

- Class is no interface and doesn't implement any interface, so principle can’t be applied directly.
- However, if were to introduce interfaces to segregate course management logic, we could make the class more flexible for various implementations (and make sure an interface is minimal and provides only that which the client needs)

```
public interface CourseOps {
        void addCourse(String courseName);
        boolean removeCourse(String courseName);
}
```

Dependency Inversion Principle (DIP)

- Class depends on concrete implementation of a `List` (`ArrayList`). Could be seen as minor violation of DIP since class is now „tied“ to the specific List implementation of `ArrayList`.
- Solution: use `List` instead of `ArrayList` for declaring the collection, allowing different list implementations to be used if needed in the constructor:

```
Import java.util.List;

public class CourseManager {
      private List<String> courses;

      public CourseManager() {
            this.courses = new ArrayList<>();
      }
      …
}
```

3)

Amal Bolakhrif (7423946), Beyza Karaca (7422370),
Lena Igoumrane (7424588)

design pattern : observer - pattern