Preliminary Design

(Component Specification)

Gerson Sunyé
University of Nantes
gerson.sunye@univ-nantes.fr
http://sunye.free.fr

Agenda

- Introduction
- Components
- Design process
- Conclusion

Introduction

Definition

- Preliminary Design is the first step of software design.

- During this phase, a high-level design concept that meets the requirement specification is created.

- The concept is expressed as a set of components with clear interfaces.

Preliminary Design Goals

- Establish the system boundaries.
- Define system and component interfaces.
- Define component scope and responsibilities.
- Specify desired component operations.

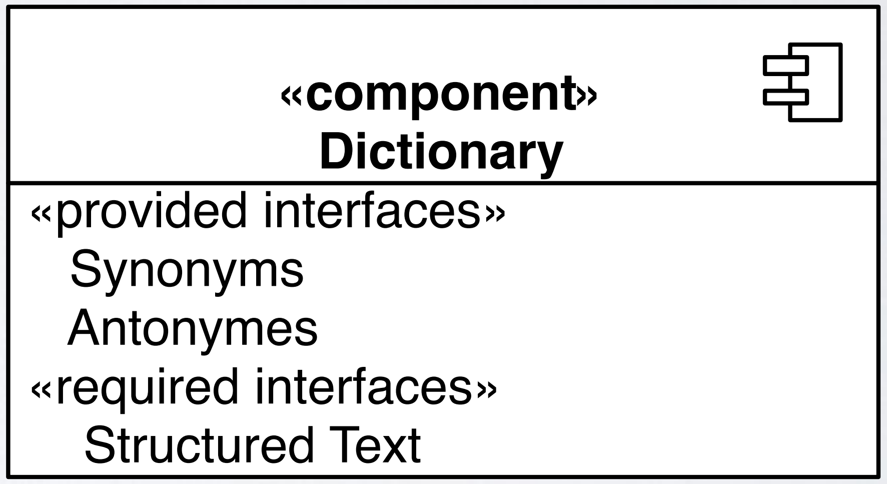
Typical Deliverables

- Component Diagram.
- Precise specification of interfaces:
  - signatures, pre and post conditions.
- Interaction diagrams.
- State machines.

Components

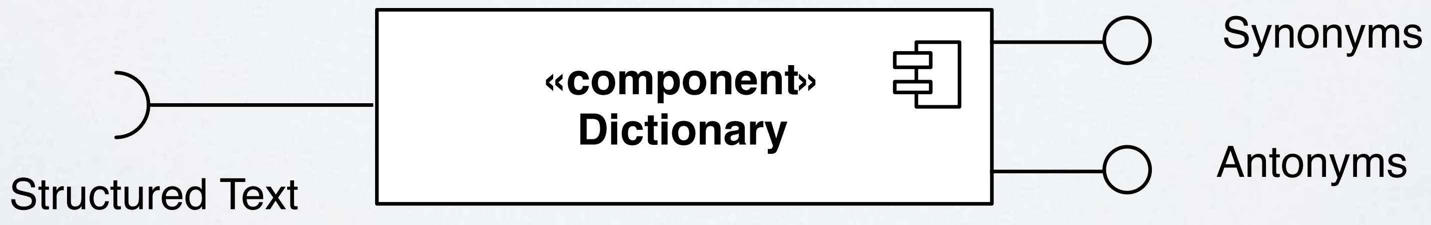
Components

- A component is a coherent package of software that can be independently developed and delivered as a unit.



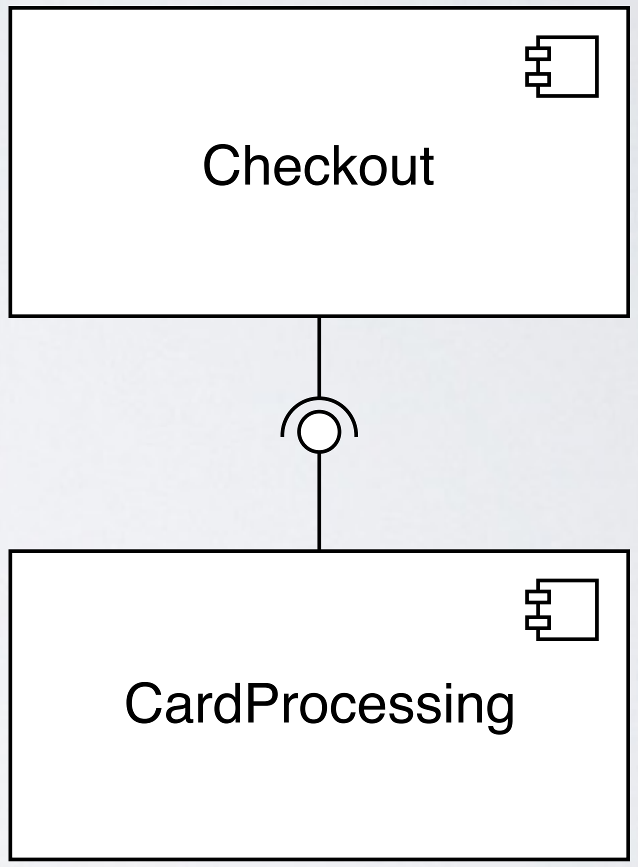
Provided and Required Interfaces

- A component has an explicit and well-specified interfaces of the:
  - provided services;
  - services expected from other components;
- Components use these interfaces to communicate with each other.



Composition

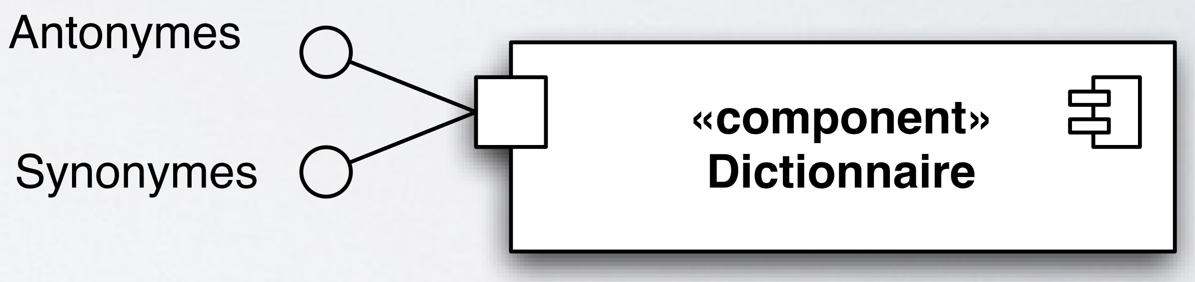
- Components can be combined with other components to provide and use services.
- Components are substitutable: one component can replace another at design time or at runtime, if the successor component meets the requirements of the initial one.



"Component-based-Software-Engineering-example1" by Cmendes at English Wikipedia

Ports assemble Interfaces

- A port represents an interaction point between a component and its environment.
- The nature of the interactions is specified by interfaces.



Benefits of Components

- Component based architectures promote:
  - Reusability and reliability.
  - Maintainability, modularity, testability, flexibility, extensibility.
  - Portability.

Design Process

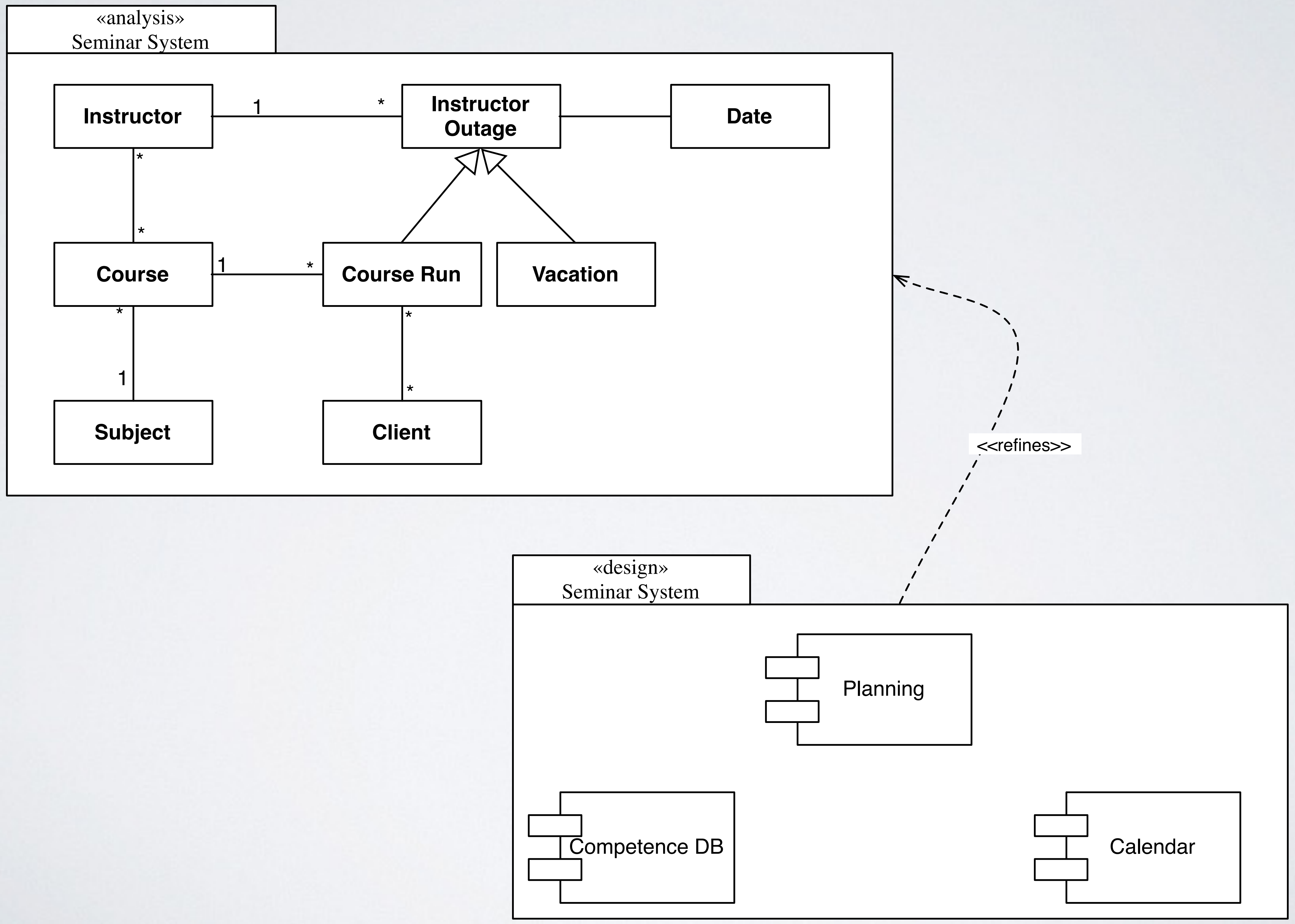
Design Process Steps

1. Adopt the domain model as the initial class model.
2. Define the boundaries:
    1. consider the system as a single component
    2. specify the system behaviour that would meet the requirements.
3. Decompose components recursively.
    1. approaches: structural and behavioral
4. Add technical components (database, user interface, middleware, etc.).
5. Use interactions to validate component interfaces.
6. Use state machines to specify classes.

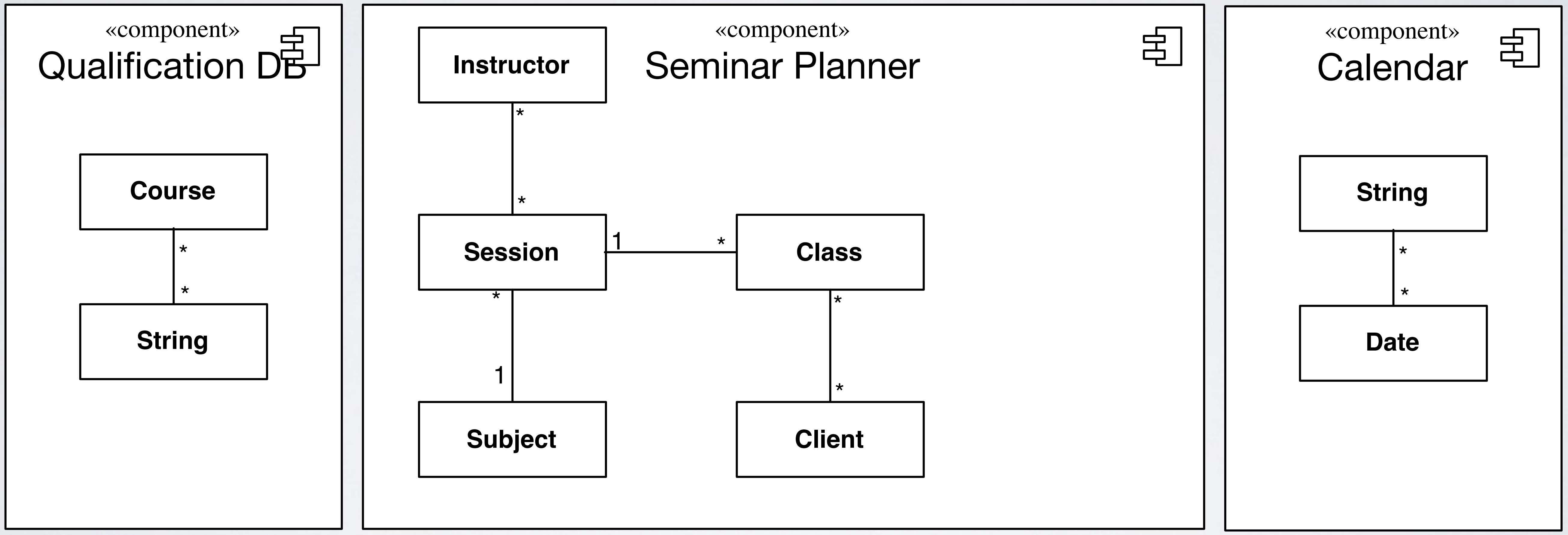
Adopt the domain model as the initial class model.

- 1-1 correspondence often not possible
- A model that gives best performance is often different from one that clearly explains what the object does.

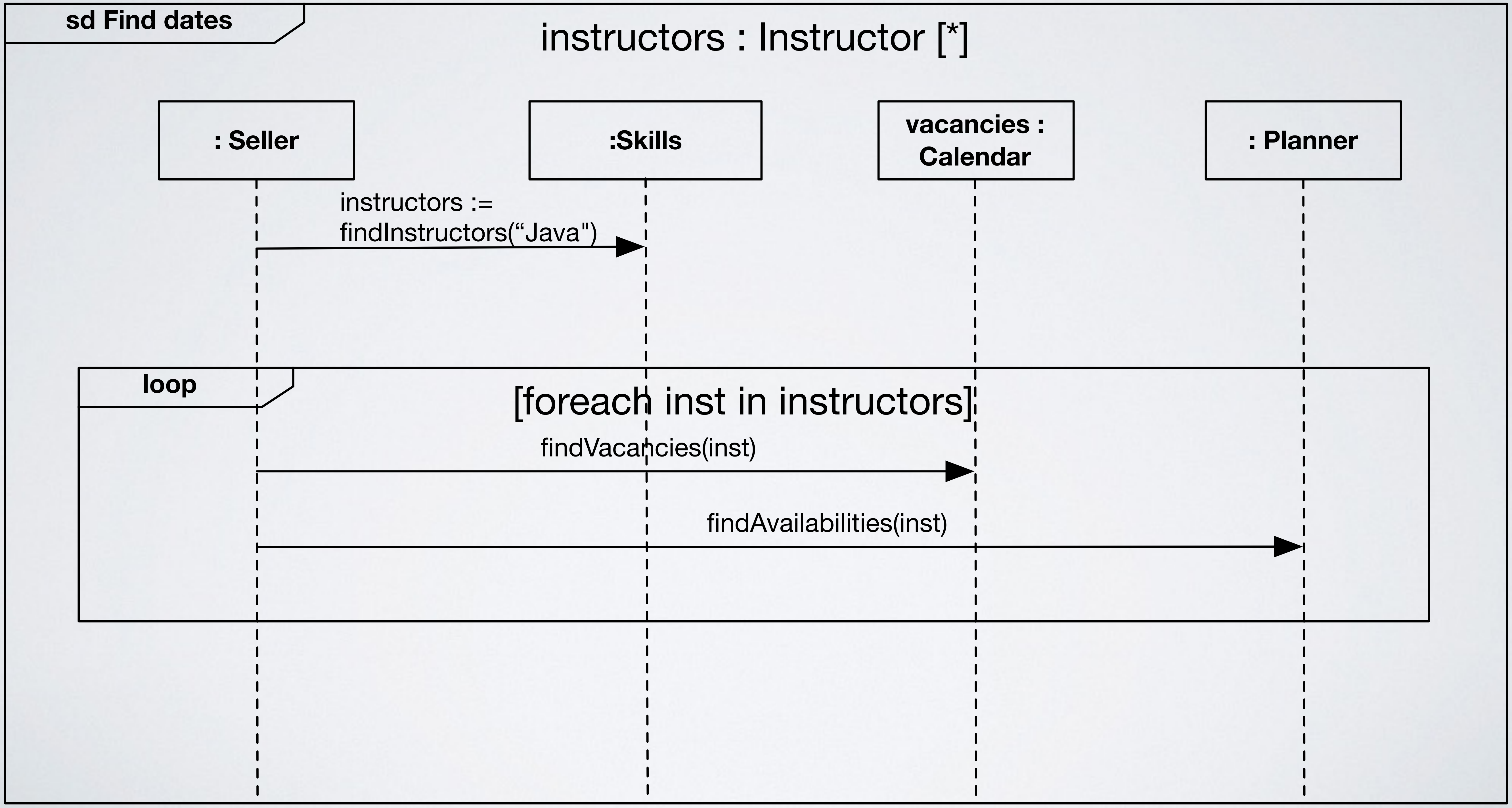
Domain Model Partition



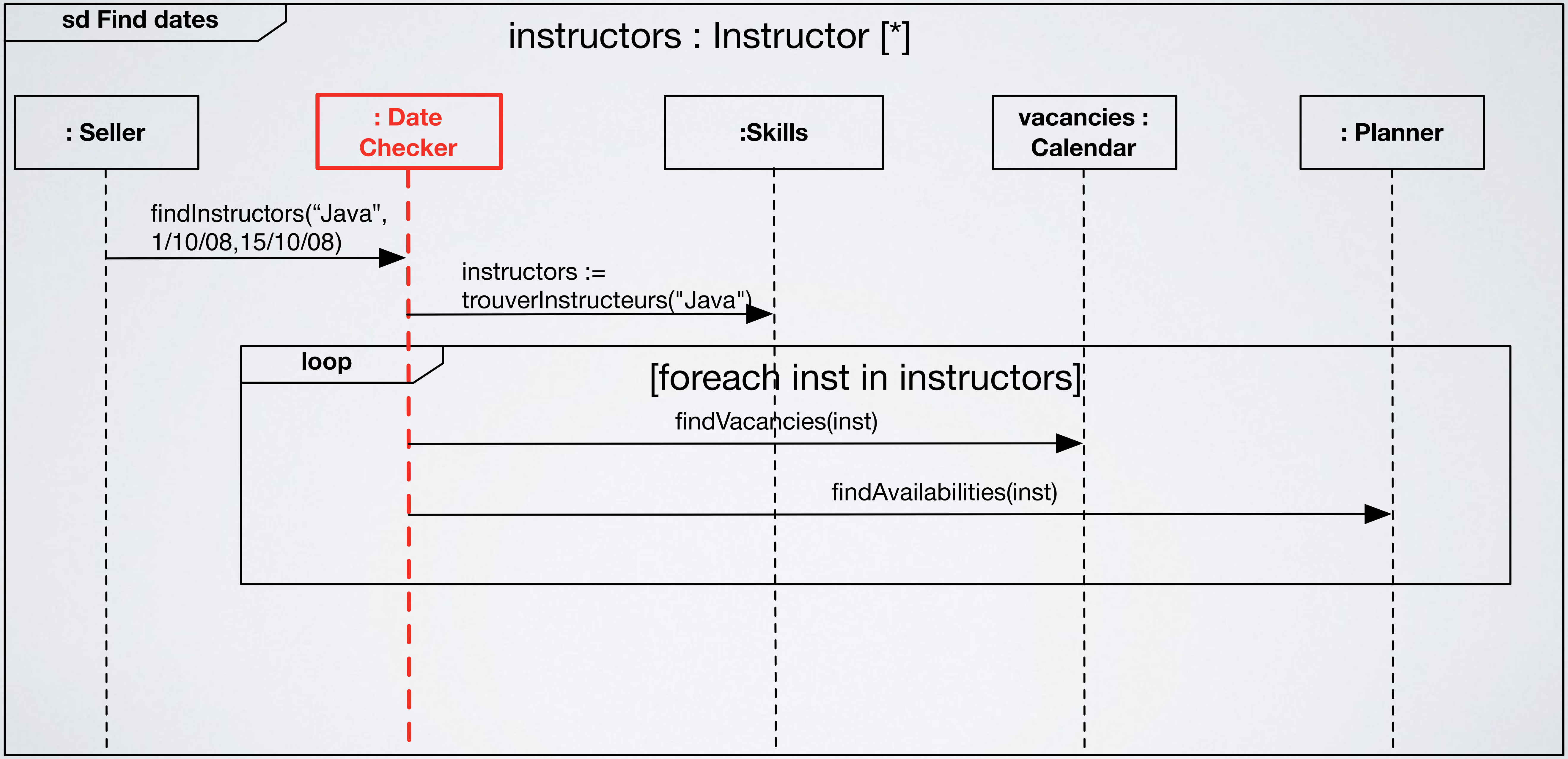
Domain Model Partition



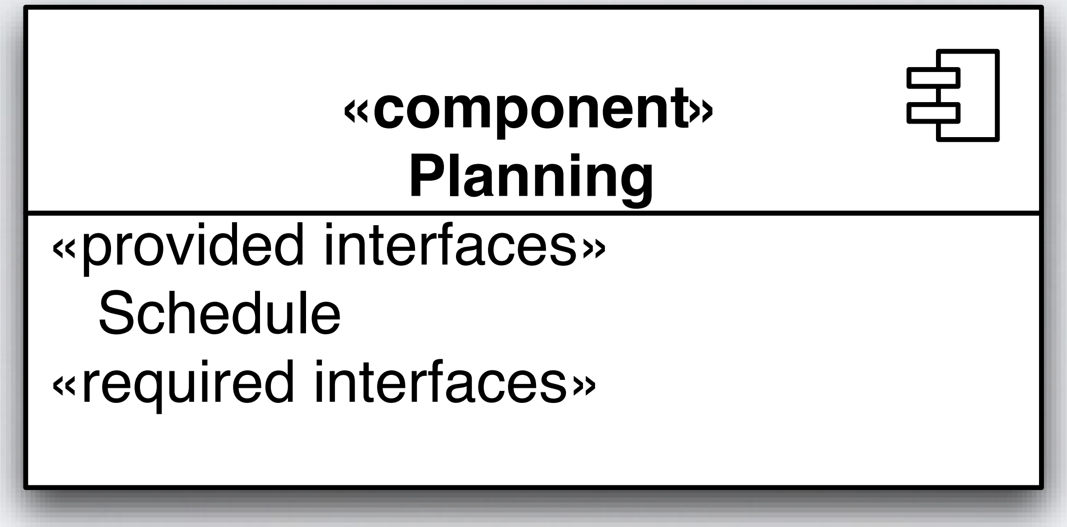
Interactions (v1)

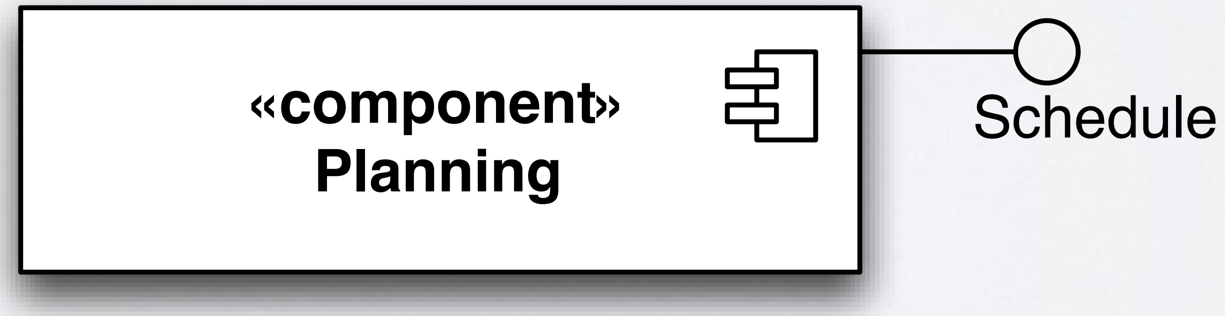


Interactions (v2)


sd Find dates
instructors : Instructor [*]
: Seller
: Date Checker
:Skills
vacancies : Calendar
: Planner
findInstructors("Java", 1/10/08,15/10/08)
instructors := trouverInstructeurs("Java")
loop
[foreach inst in instructors]
findVacancies(inst)
findAvailabilities(inst)

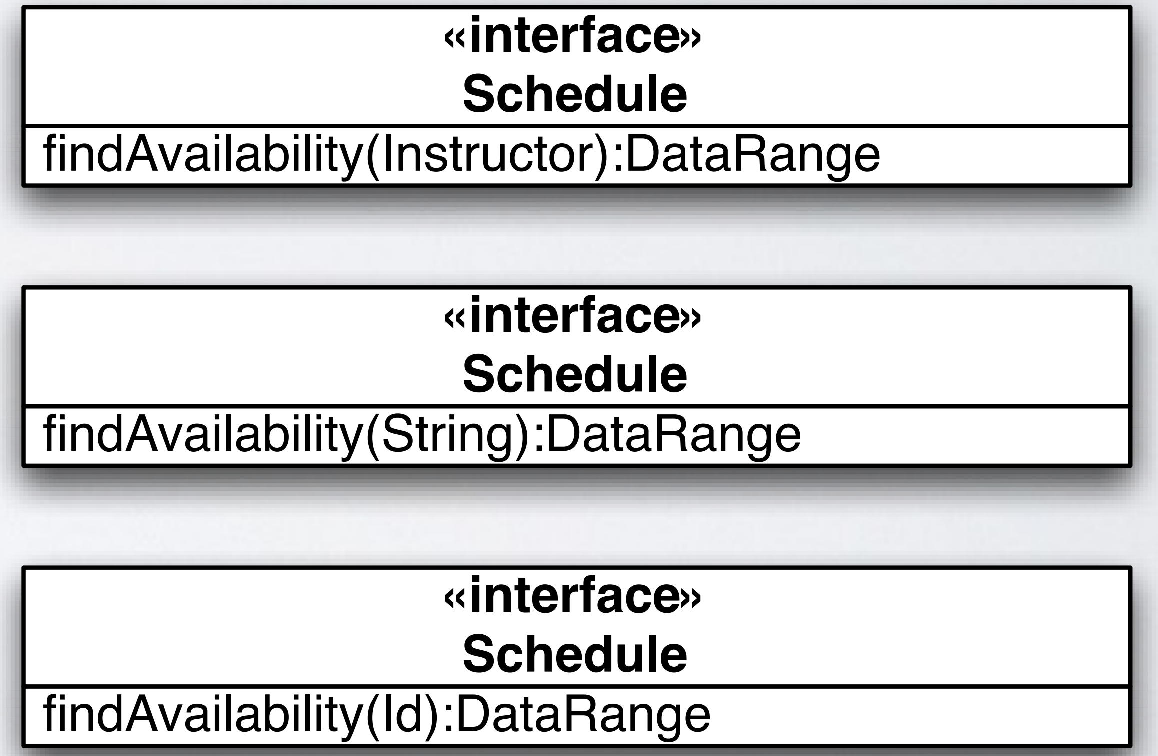
Interface Specification



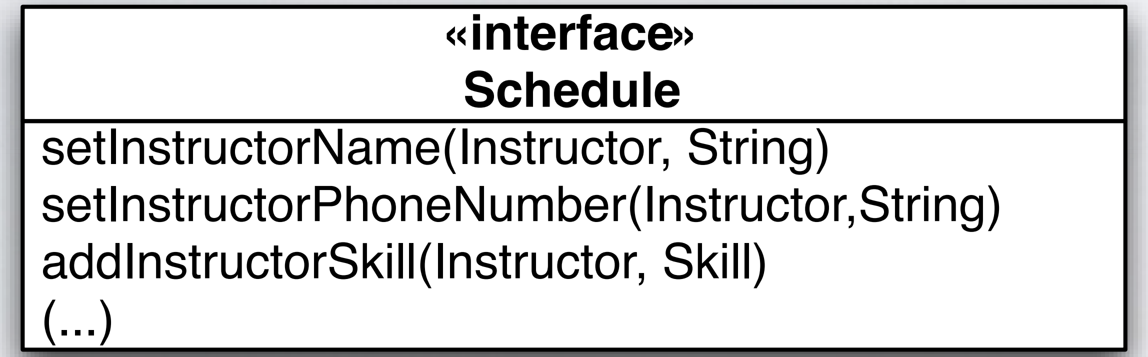


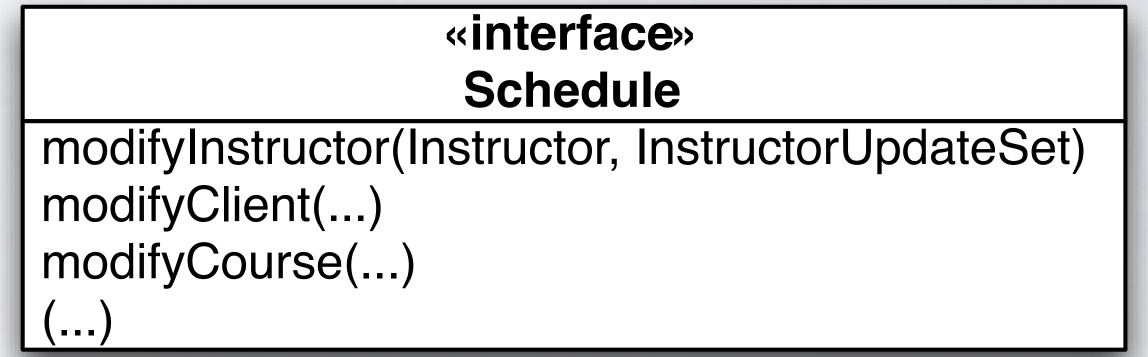
Parameter Type Specification

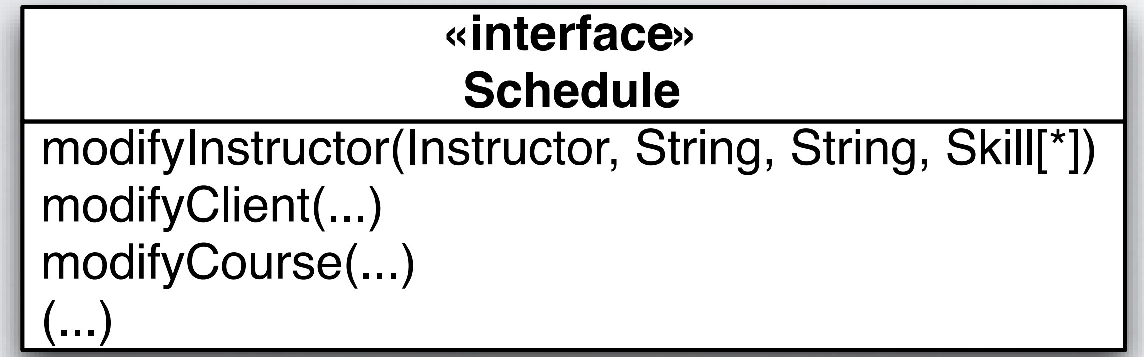
- Internal classes
- Primitive types
- Datatypes

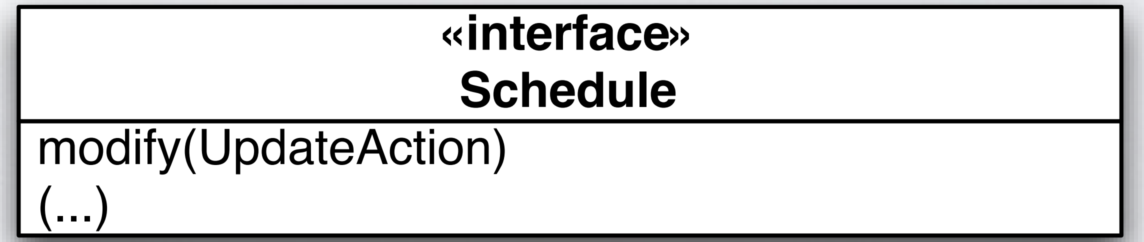

«interface»
Schedule
findAvailability(Instructor):DataRange
«interface»
Schedule
findAvailability(String):DataRange
«interface»
Schedule
findAvailability(Id):DataRange

Operation Specification

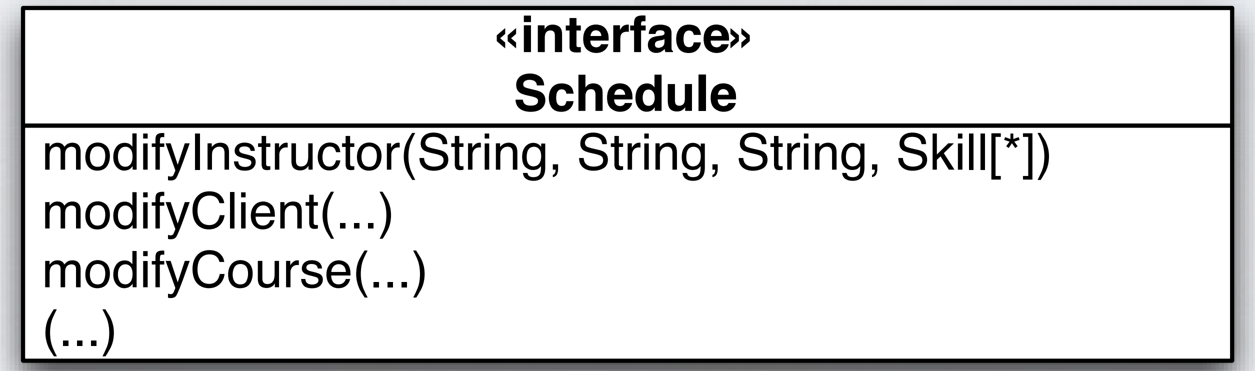

«interface»
Schedule
setInstructorName(Instructor, String)
setInstructorPhoneNumber(Instructor,String)
addInstructorSkill(Instructor, Skill)
(...)

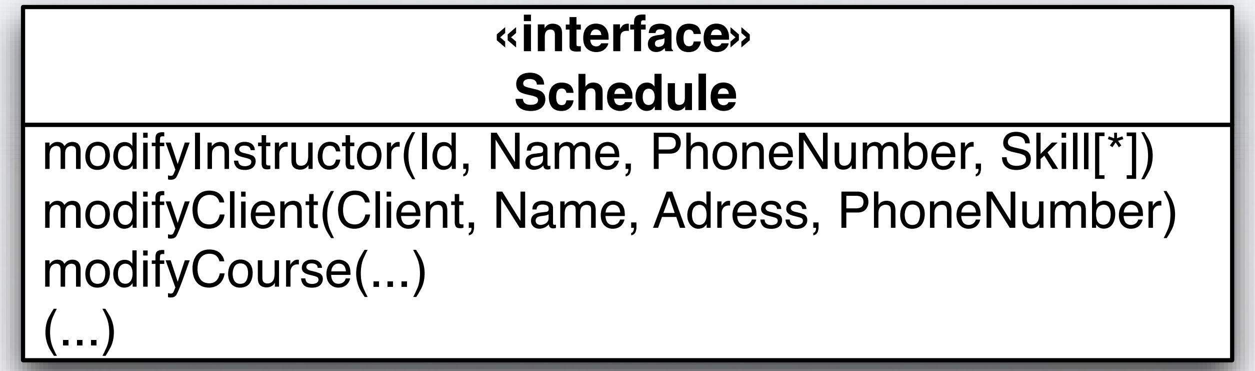






Parameter Types




«interface»
Schedule
modifyInstructor(Id, Name, PhoneNumber, Skill[*])
modifyClient(Client, Name, Adress, PhoneNumber)
modifyCourse(...)
(...)

Precise Specification of Operations

```
modifyInstructor(instructor:String, name:String, phone: String)

pre: instructor.size() > 0 and (...)

pre: instructor.notEmpty() and (...)

pre: self.instructors->exists(id = instructor) and (...)
```

Conclusion

Conclusion

- Component partitioning requires several iterations:
  - it's hard to find the adequate partitioning at first time.
  - design experience is required.
- Interfaces should be designed to be stable.
  - Good APIs do not "appear", they must be designed.

References

- Objects, components, and frameworks with UML: the Catalysis Approach, by Desmond D'Souza and Alan Wills. Addison Wesley, 1998.
- Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. Craig Larman. Prentice Hall, 2004.
- UML Distilled: A Brief Guide to the Standard Object Modeling Language (3rd Edition). Martin Fowler. Addison-Wesley Professional, 2003.
- Patterns of Enterprise Application Architecture. Martin Fowler. Addison-Wesley Professional, 2002.

Additional Readings

- Design patterns:
  - Command.
  - Façade.
  - Data Transfer Object (DTO).