DPF EXCERCISE #8

TUD, ST Group
Dr. Sebastian Götz

Role Modelling

The Horse Show

A Role Modelling Example

Task #1 – The Horse Show

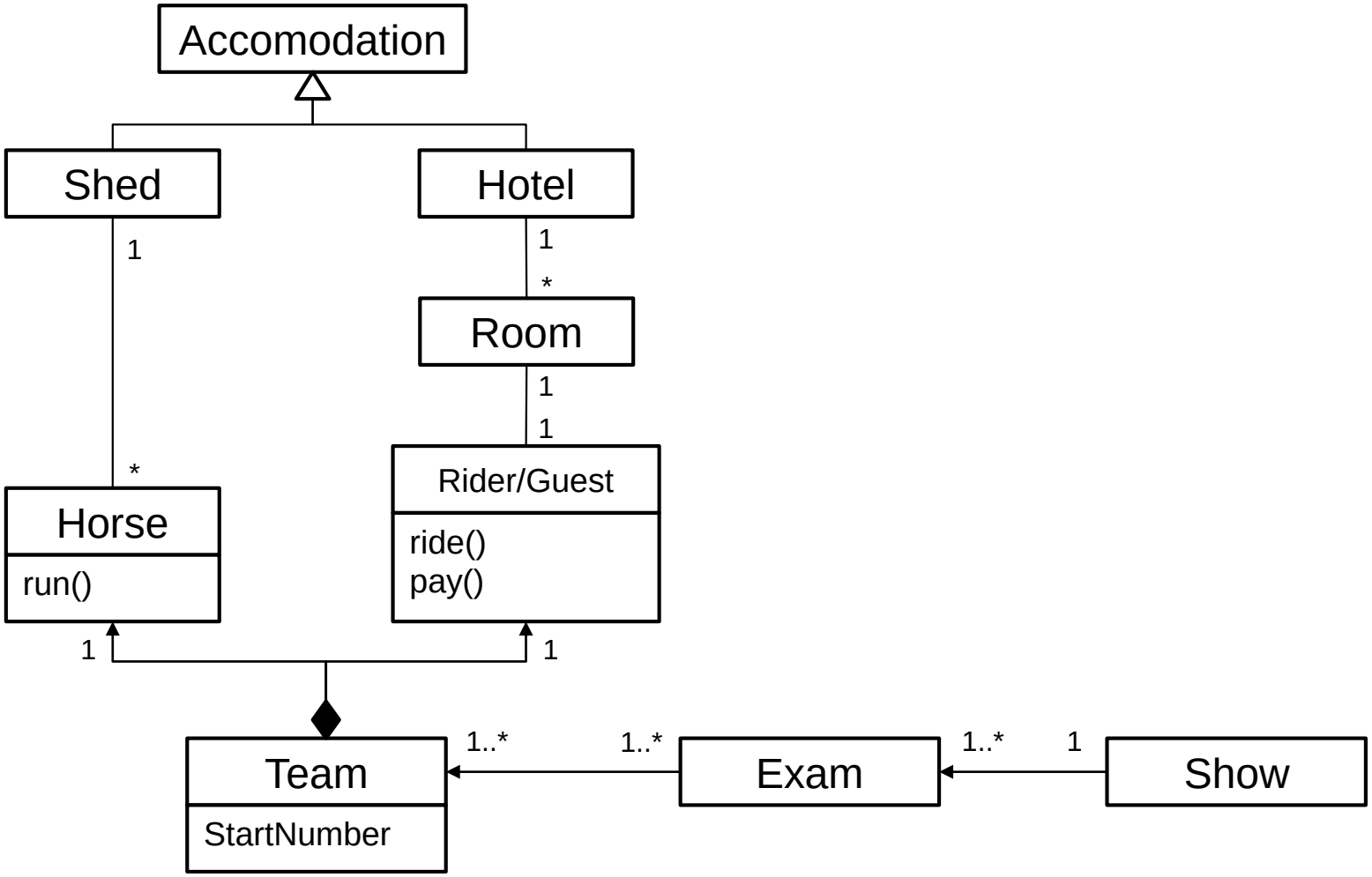
You are to develop software for the management of **horse shows**.

In the world of horse shows, there are **horses** and **riders** who *together obtain a starting number*, with which they can *inscribe* for **examinations**, which are managed by **referees**.

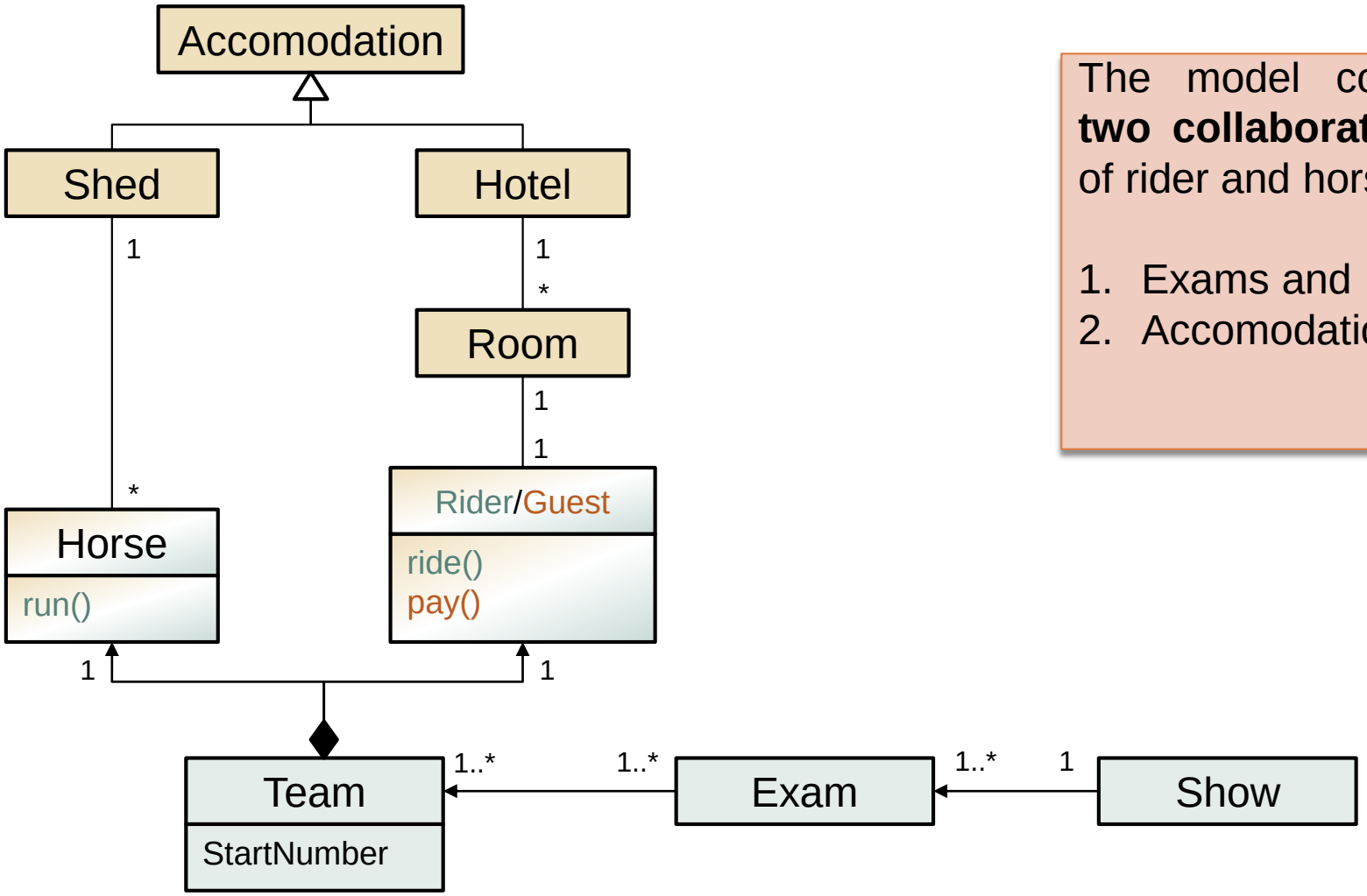
Of course, because the average horse show takes about two to three days, *both horses and riders need a place to stay*. It is one of the tasks of the organization team to provide ***accommodation for horses and riders***.

1. Draw an object-oriented design model.
2. Identify problems of this design.
3. Draw a role-oriented design model.

Task #1 – The Horse Show



Task #1 – The Horse Show



The model covers **two collaborations** of rider and horse:

1. Exams and
2. Accomodation.

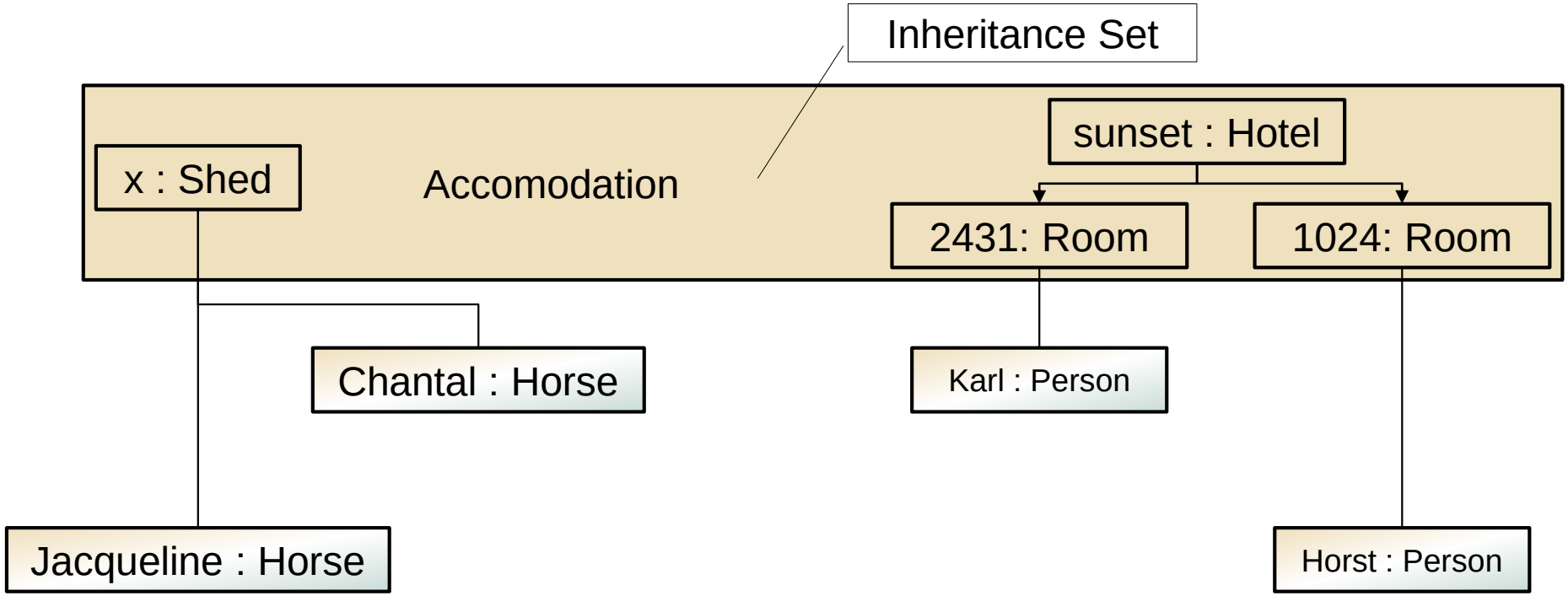
Task #1 – The Horse Show

- Problems of this solution
  - Classic SE problems
    - Extensibility
    - Variability
    - Maintainability
  - Specific Problem: Lack of knowledge expressed
    - Many aspects of the domain are not covered
    - Required for a working MDA!

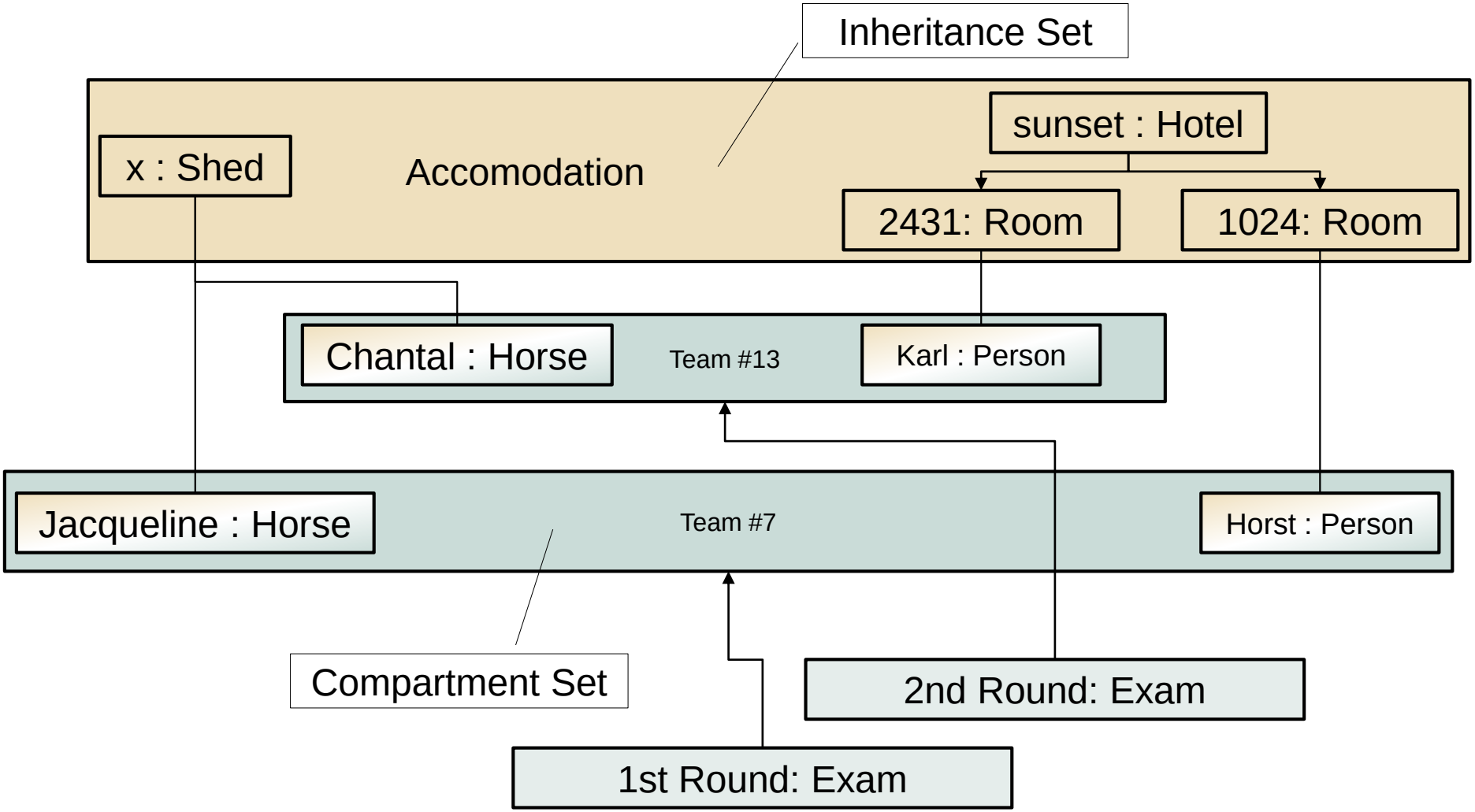
Task #1 – The Horse Show

- Class vs. Object Model
  - **Inheritance** denotes a subset relation ($\subset$)
    - Stable $\subset$ Accomodation
    - Hotel $\subset$ Accomodation
  - **Instance-of** denotes an element-of relation ($\in$)
    - „Sunset Hotel, 3820 NY“ $\in$ Hotel
    - „Sunset Hotel, 3820 NY“ $\in$ Accomodation
    - In OO, objects have an identity!
  - **Compartments** (part-of) denote complex objects (sets)
    - They define a **special set** of objects
    - The **whole** depends existencially on its **parts**
    - The identity of the whole is composed of the identity of its parts
      - There is no team without a horse and a rider
      - But, horse and rider can exist without being a team

Task #1 – The Horse Show



Task #1 – The Horse Show



Task #1 – The Horse Show

The system we model describes at least **two collaborations**:

1. **Accomodation**: Persons and Horses staying in a hotel or shed
2. **Examinations**: Persons with Horses participate in examinations

The objects in our diagram take part in these collaborations at different times.

Problem: Code for both collaborations will be intertwined (*tangled*)

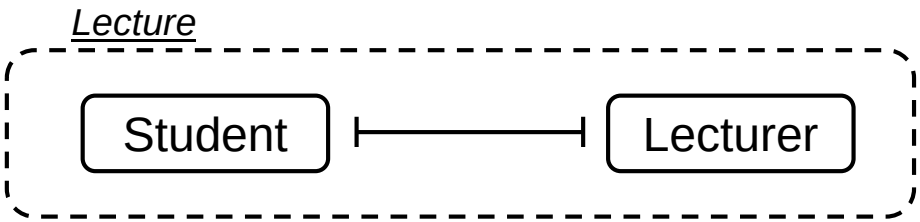
➔ bad extensibility, maintainability, etc.

Goal: Separation of Concerns (here collaborations)

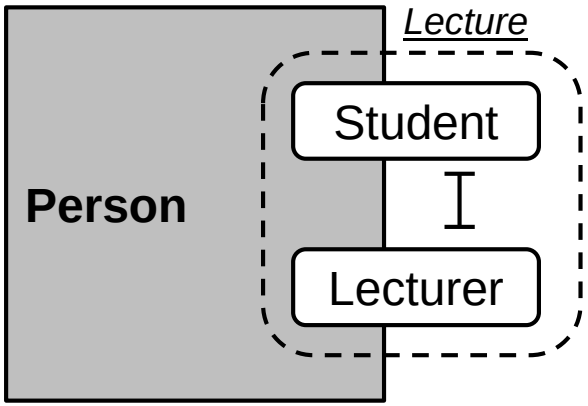
➔ **Collaborations can be formalized by role models!**

Task #1 – The Horse Show

- Role models describe **collaborations**

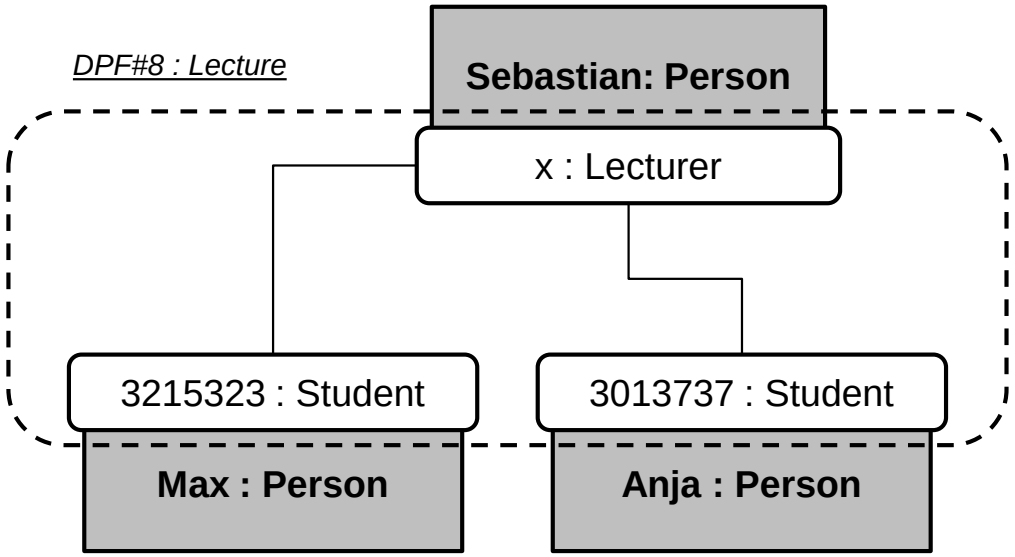


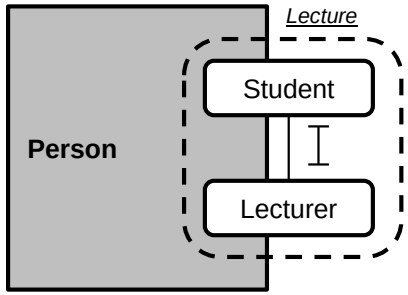
- Role models describe role types!
- Class-role models describe the mapping of role models to class-models
- Role constraints (D. Riehle):
  - Role-use
  - Role-prohibition
  - Role-implication
  - Role-equivalence



Task #1 – The Horse Show

- **Object diagram**





- Role types ⇔ Roles
- Classes ⇔ Objects
- Role models ⇔ Collaborations

Task #1 – The Horse Show

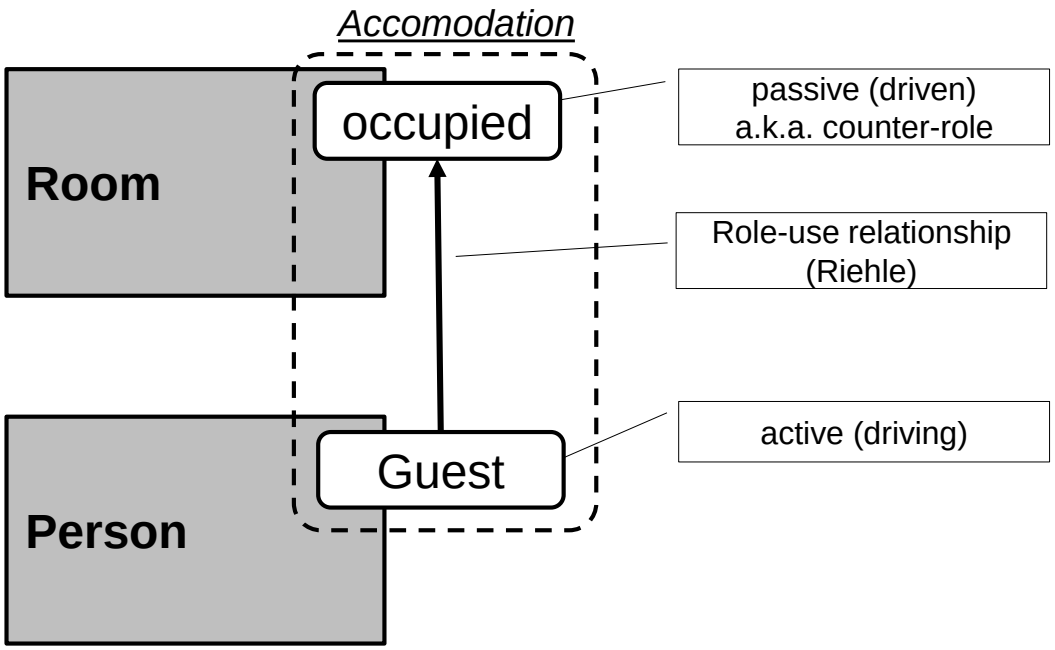
- **Role Types** are **non-rigid** and **founded** types!
  - **Non-rigid:** instances *can loose the type* without loosing their identity
    - **Example:** Student, Lecturer, Employee, etc.
  - **Founded types** always depend on another type
    - **Example**: Reader ⇔ Book, Friend ⇔ Friend, Speaker ⇔ Listener, etc.
    - Hence, role types are always part of a **collaboration**!
- **Natural Types** are rigid and non-founded
  - **Example**: Tree, Person, Animal
- **Role models** describe relations between role types, i.e., types of collaborations.
- Instances of role types (i.e., roles) are **played by** instances of natural types (i.e., objects)
- **Speciality**: **compartments** are collaborations, too!

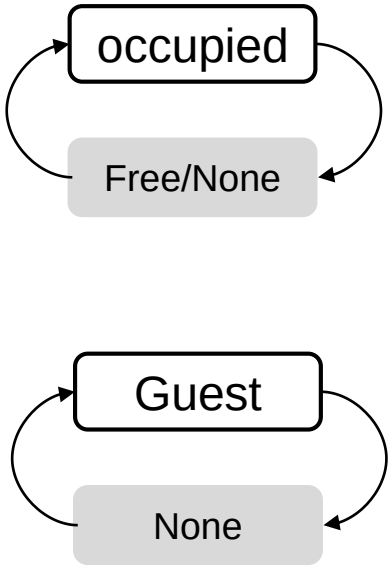
Task #1 – The Horse Show

- Thus, in our example, we have 3 collaborations:
  - Accomodation
  - Examinations
  - Compartment of Rider and Horse

Task #1 – The Horse Show

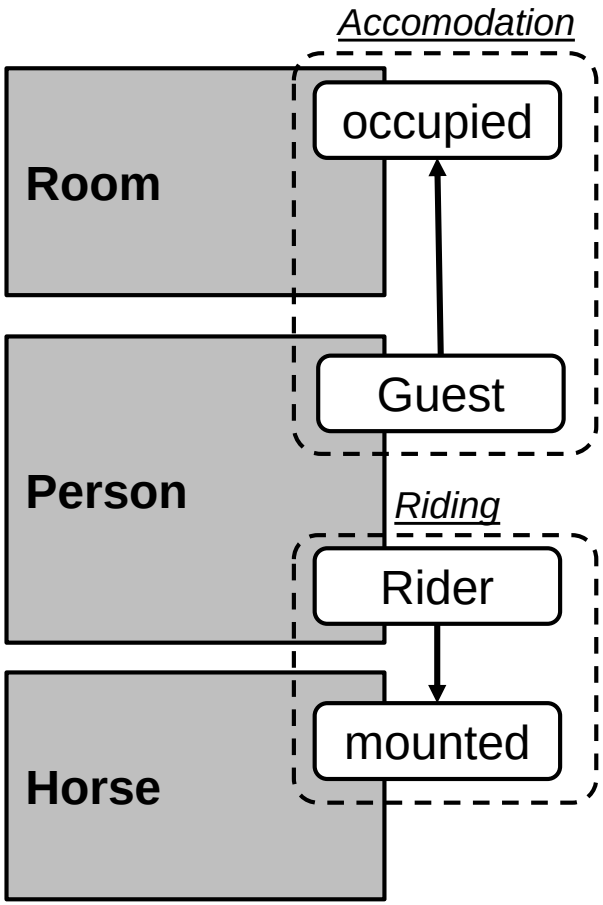
Natural Types | **Role Types** | **Collaboration Type(s)** | **Role Play Automata**

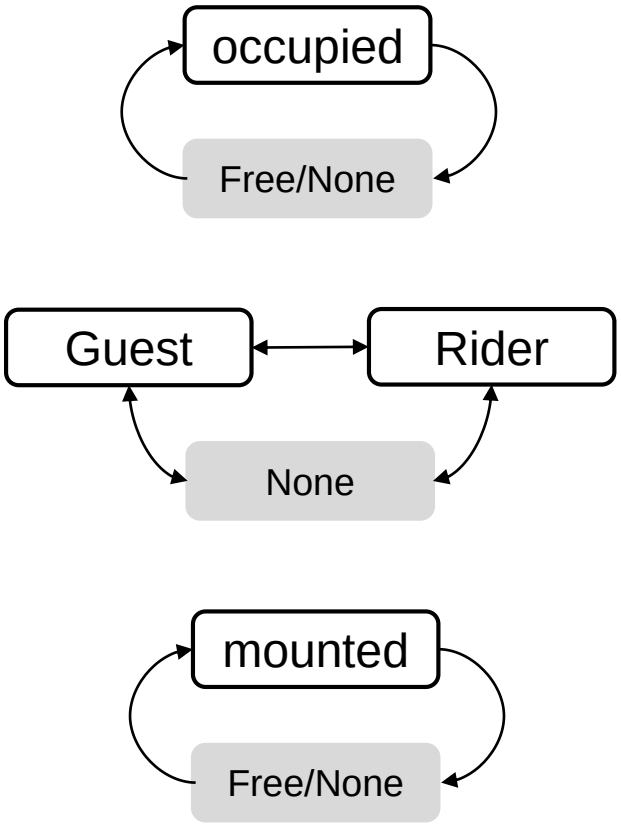




Task #1 – The Horse Show

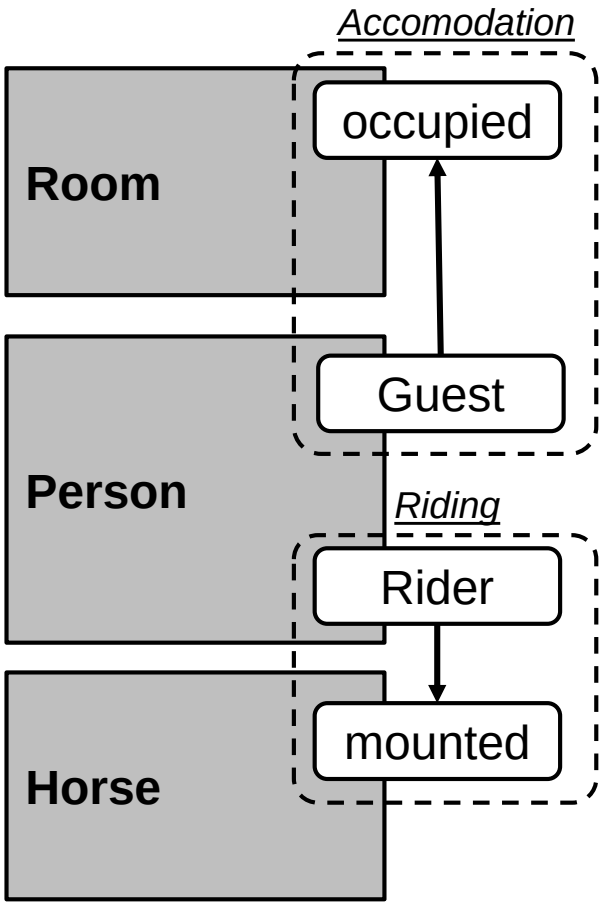
Natural Types | **Role Types** | **Collaboration Type(s)** | **Role Play Automata**





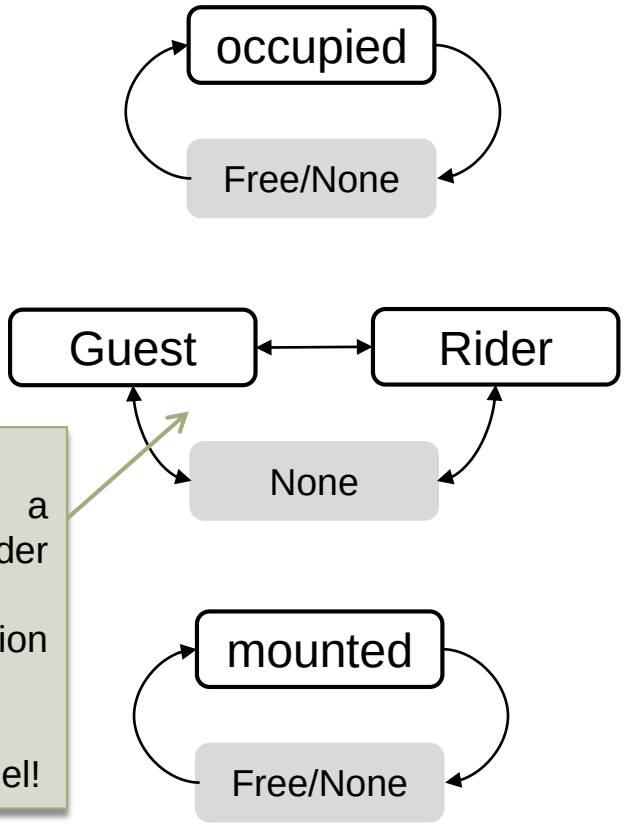
Task #1 – The Horse Show

Natural Types | **Role Types** | **Collaboration Type(s)** | **Role Play Automata**

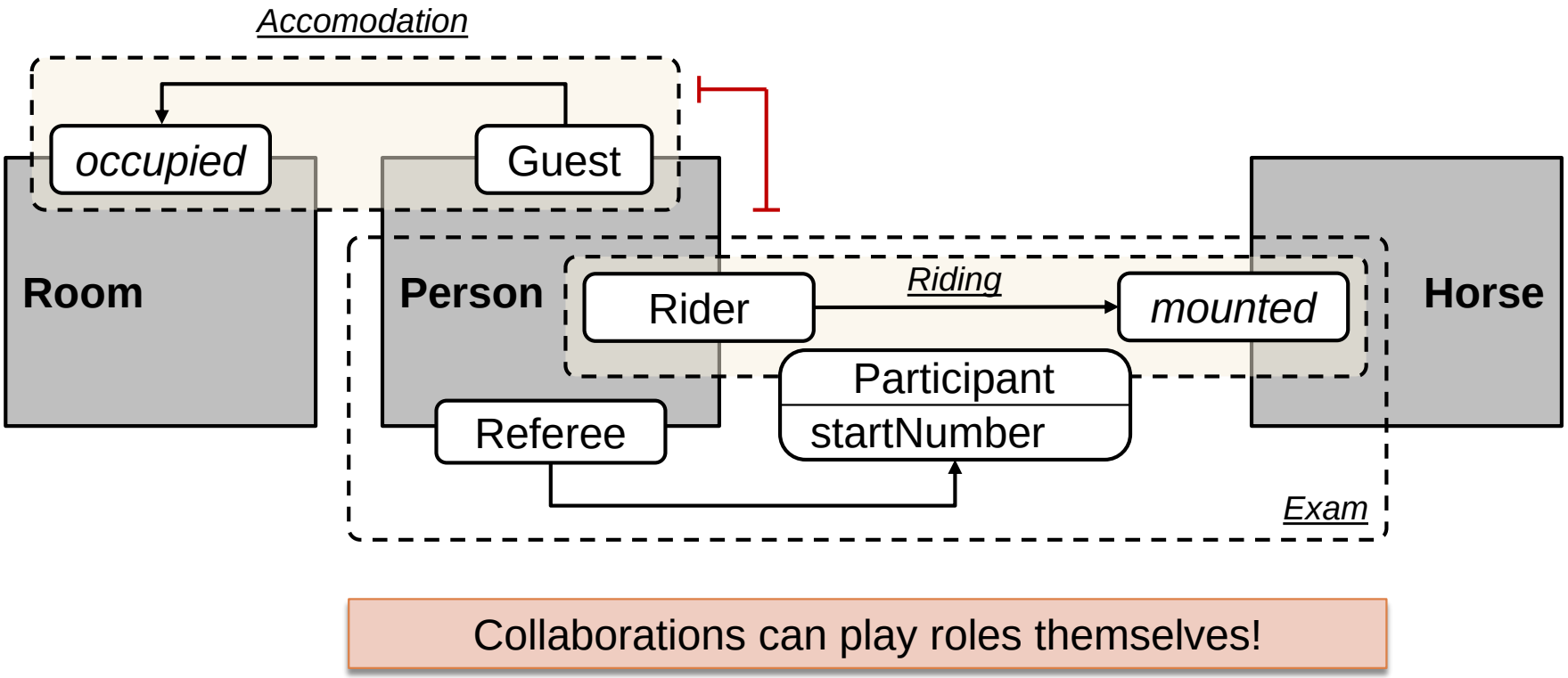


Problem:
How to specify that a guest must not be a rider at the same time?
(i.e., inter-collab-oration constraints)

➔ Enclosing Role Model!



Task #1 – The Horse Show

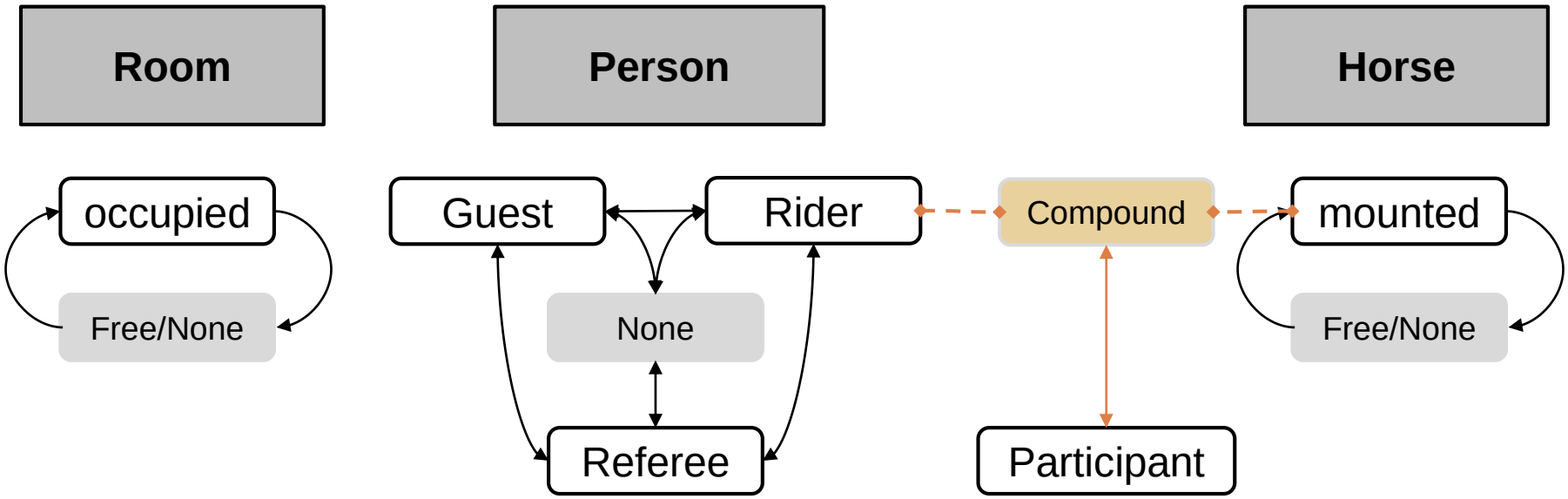


Collaborations can play roles themselves!

Are role models rigid ? ➔ **No**, horse and rider can detach without ceising to exist.
Are role models founded ? ➔ **No**, only via role types.

Task #1 – The Horse Show

Role Play Automata:



Task #1 – The Horse Show

- What have we gained (besides more complexity) ?
  - Separation of Concerns!
  - Developers can independently work on all three collaborations.
    - Imagine new types of hotels, payment, etc.
    - Imagine new types of examinations (artistic rider on two horses, …)
  - More information for code generation (MDA)

Rigidity and Foundedness

Identity ⇔ Type dependency \ Type ⇔ Type dependency	**Non-founded** Type does not depend on other types.	**Founded** Type depends on other types.
Rigid Identity depends on type.	**Natural types** (OO classes)	*bidirectional existencial compartments*
Non-rigid Identity does not depend on type.	**Contexts** (Collaboration Types, Role models)	**Role types**

Role Models of Design Patterns

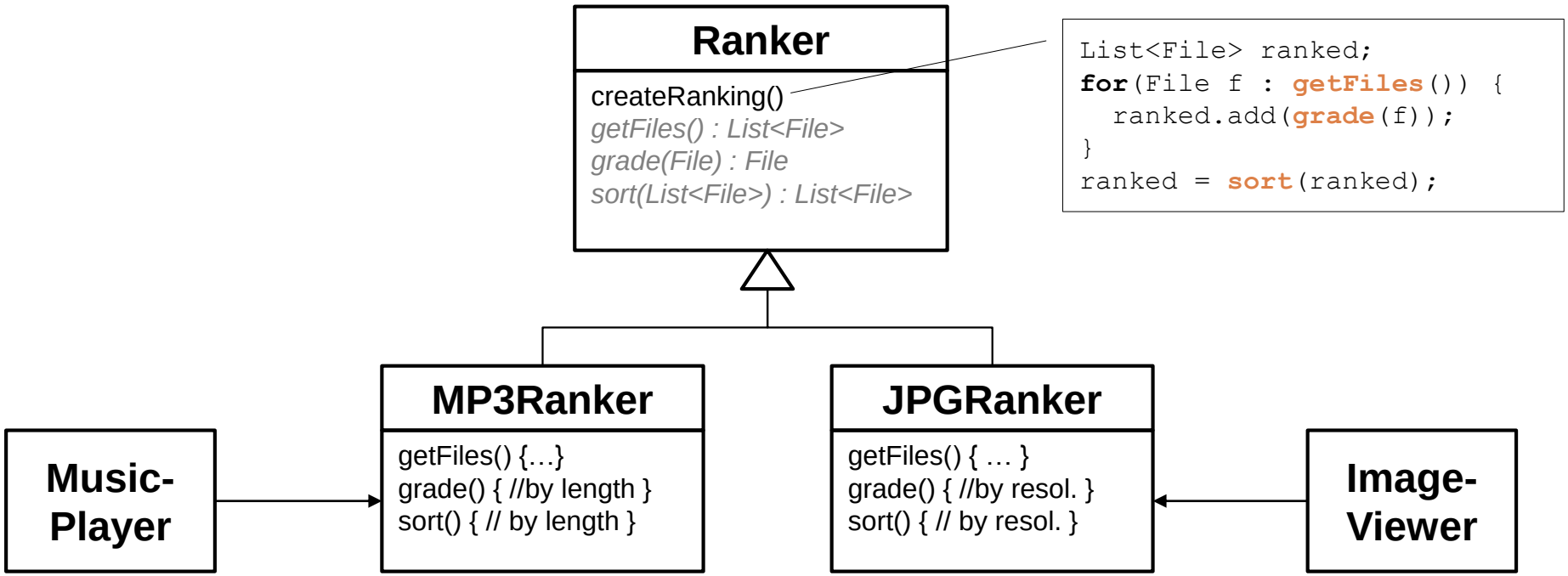
Template Method

Template Class

Dimensional Class Hierarchies

Task #2 – Template Method

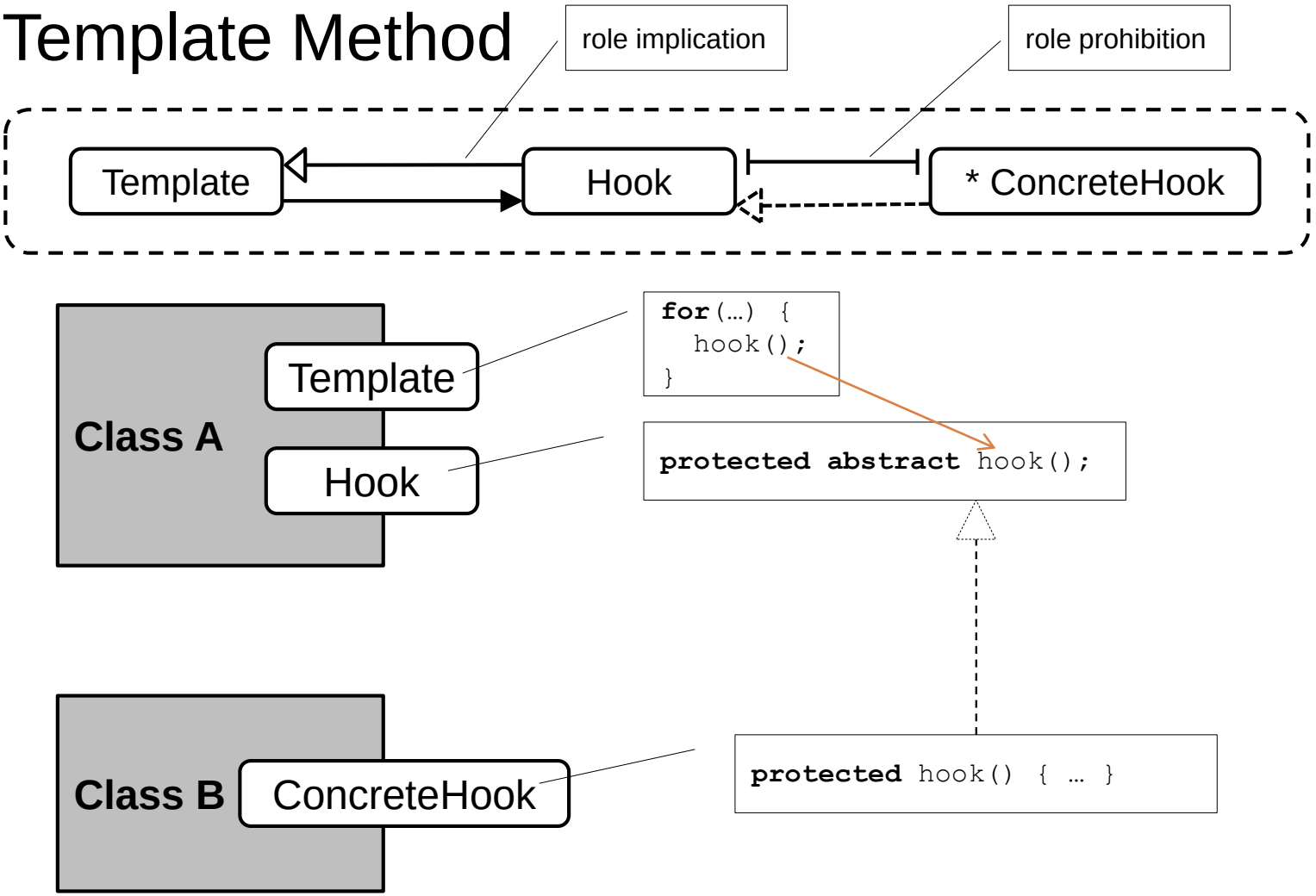
- Example: File Ranking Framework



- Subclasses as framework instances

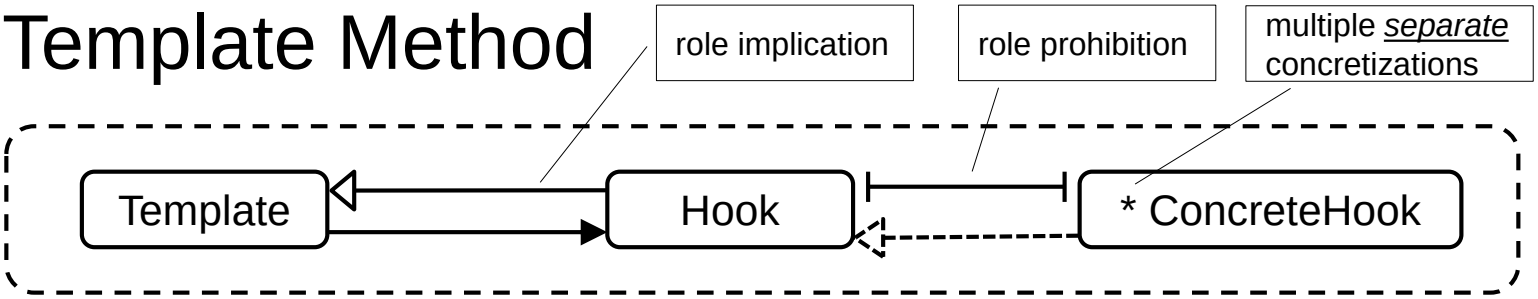
Task #2 – Template Method

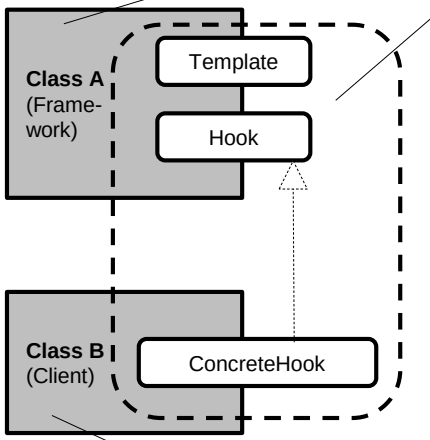
- Template Method



Task #2 – Template Method

- Template Method





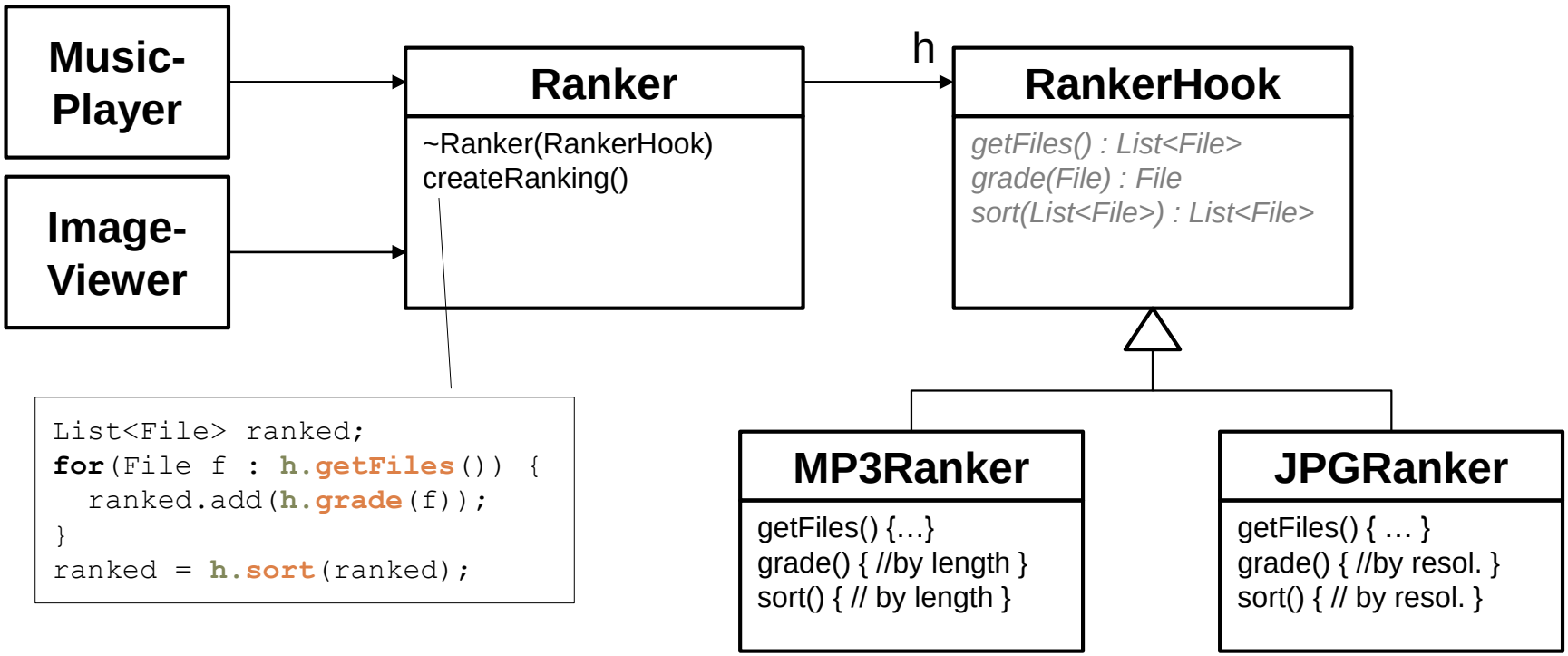

```
class A {…}
```

```
team class TemplateMethod {
  role Template playedBy A {
     void templateMethod() {
       //returns most specific sub-role of Hook
       Hook h = base.getRole(Hook.class)
       for(…) { h.hook(); } }
  }
  role Hook playedBy A {
     protected abstract void hook();
  }
  role ConcreteHook extends Hook playedBy B {
     protected void hook() { … }
  }
}
```

```
class B {…}
```

Task #2 – Template Class

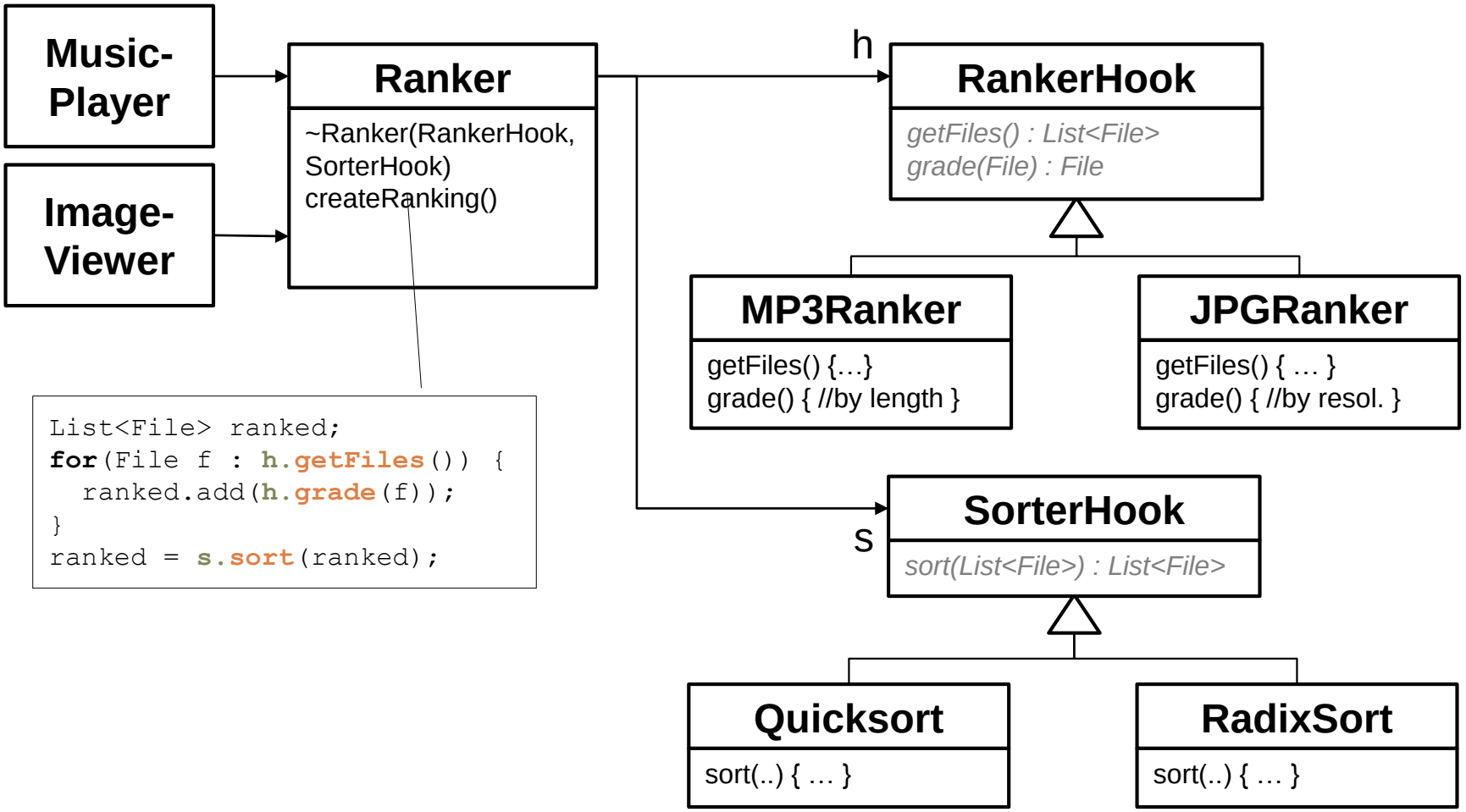
- Example: File Ranking Framework



- Subclasses as framework instances

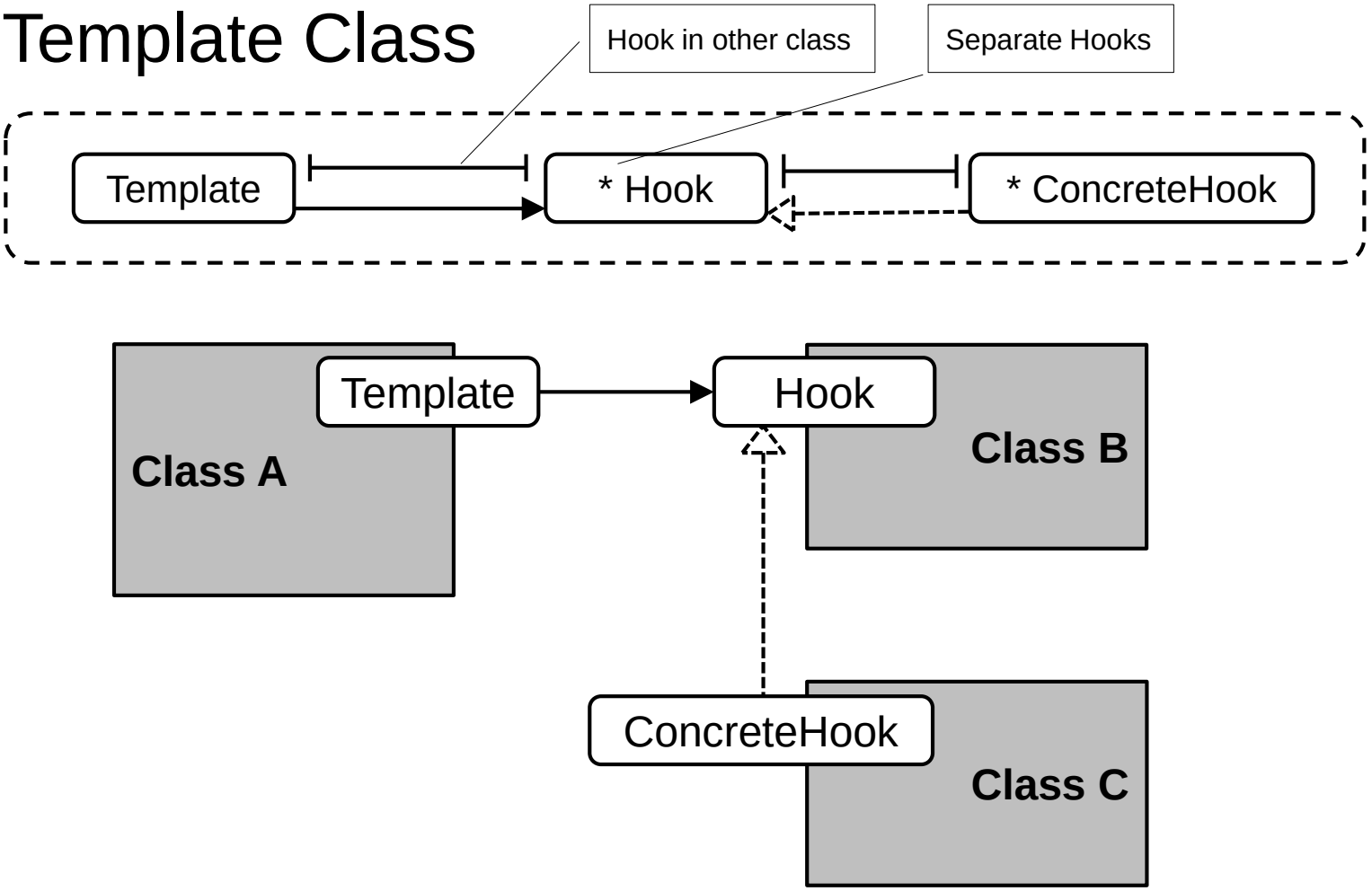
Task #2 – Template Class

- **Incentive**: vary hooks independent from template (esp. for multiple hooks)



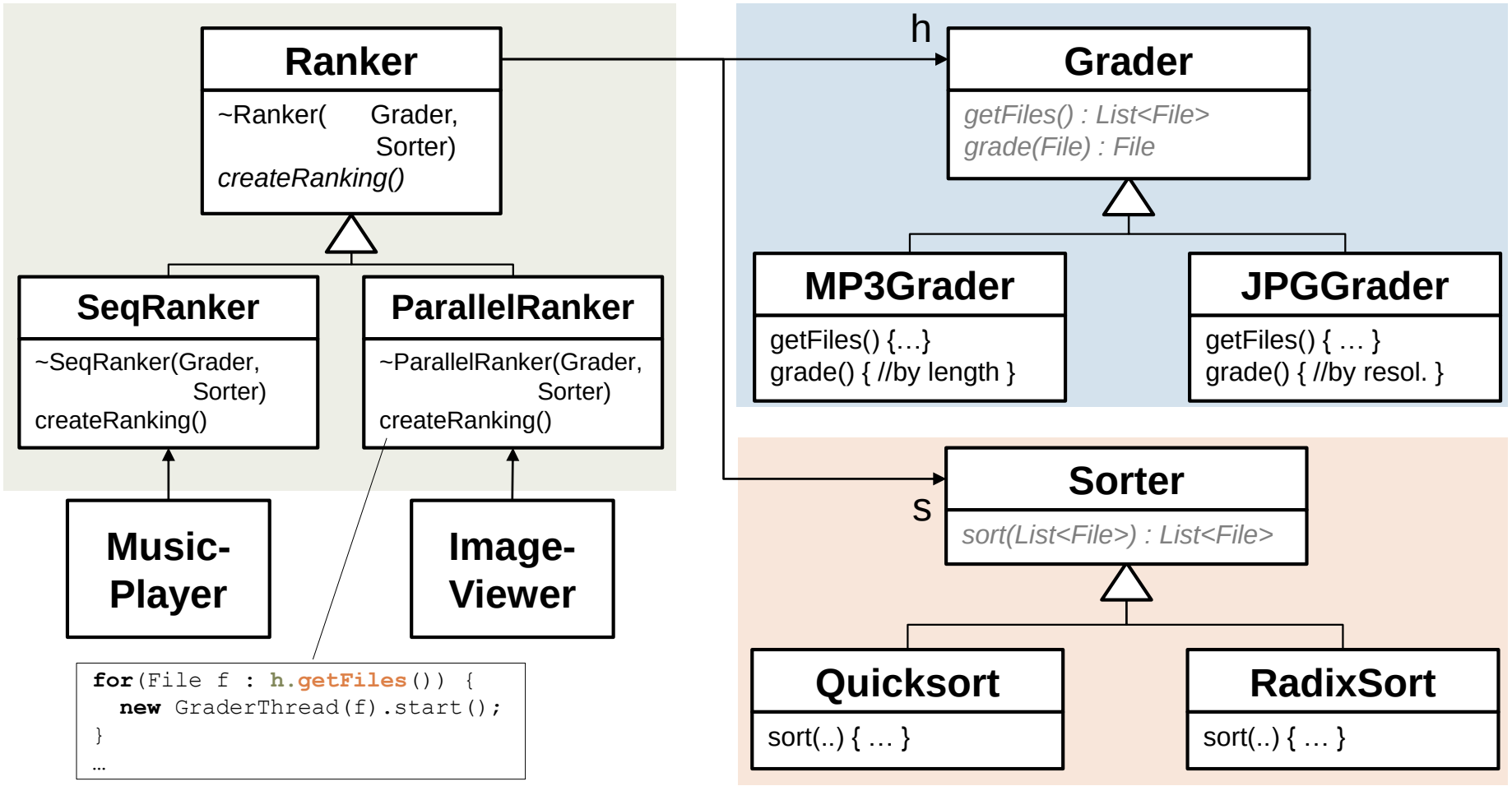
Task #2 – Template Class

- Template Class



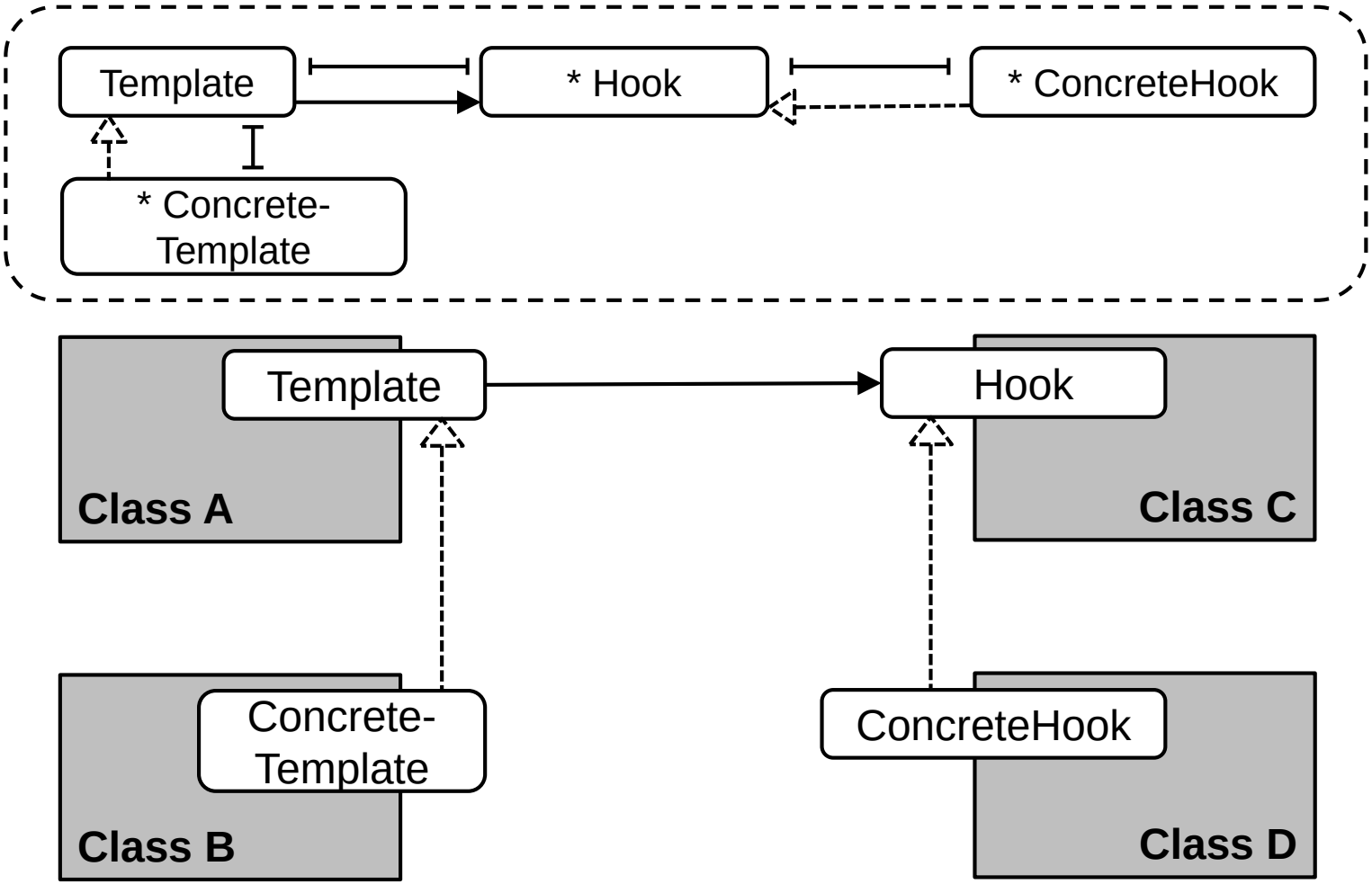
Task #2 – Dimensional Class Hierarchies

- **Incentive**: vary hooks independent from template (esp. for multiple hooks)



Task #2 – Dimensional Class Hierarchies

- **Incentive**: vary template indepently, too



Application Domains for Roles

- Where is role-oriented modelling and programming most beneficial?
  - Games
    - RPGs, Sports (Racing, Soccer), Shooter, etc.
    - Games are about collaborations between the player and computer-controlled characters!
  - Administration tools (SLM, CRM, etc.)
    - Including graphical user interfaces
    - Domains often inherently describe collaborations!
  - Self-adaptive systems
    - Systems adjust themselves to changes in their environment
    - Thus, they describe collaborations between the environment and themselves!

Interested?

- Graduate College: **RoSI** (Role-oriented System Infrastructures)
  - Basic research on a complete software development process using roles
- Collaborative Research Center: **HAEC** (Highly-adaptive Energy-efficient Computing)
  - Research on self-adaptive software optimizing for energy-efficiency

- Interested for your thesis or SHK (student job)?
  - Ask me or send a mail! (sebastian.goetz@acm.org)