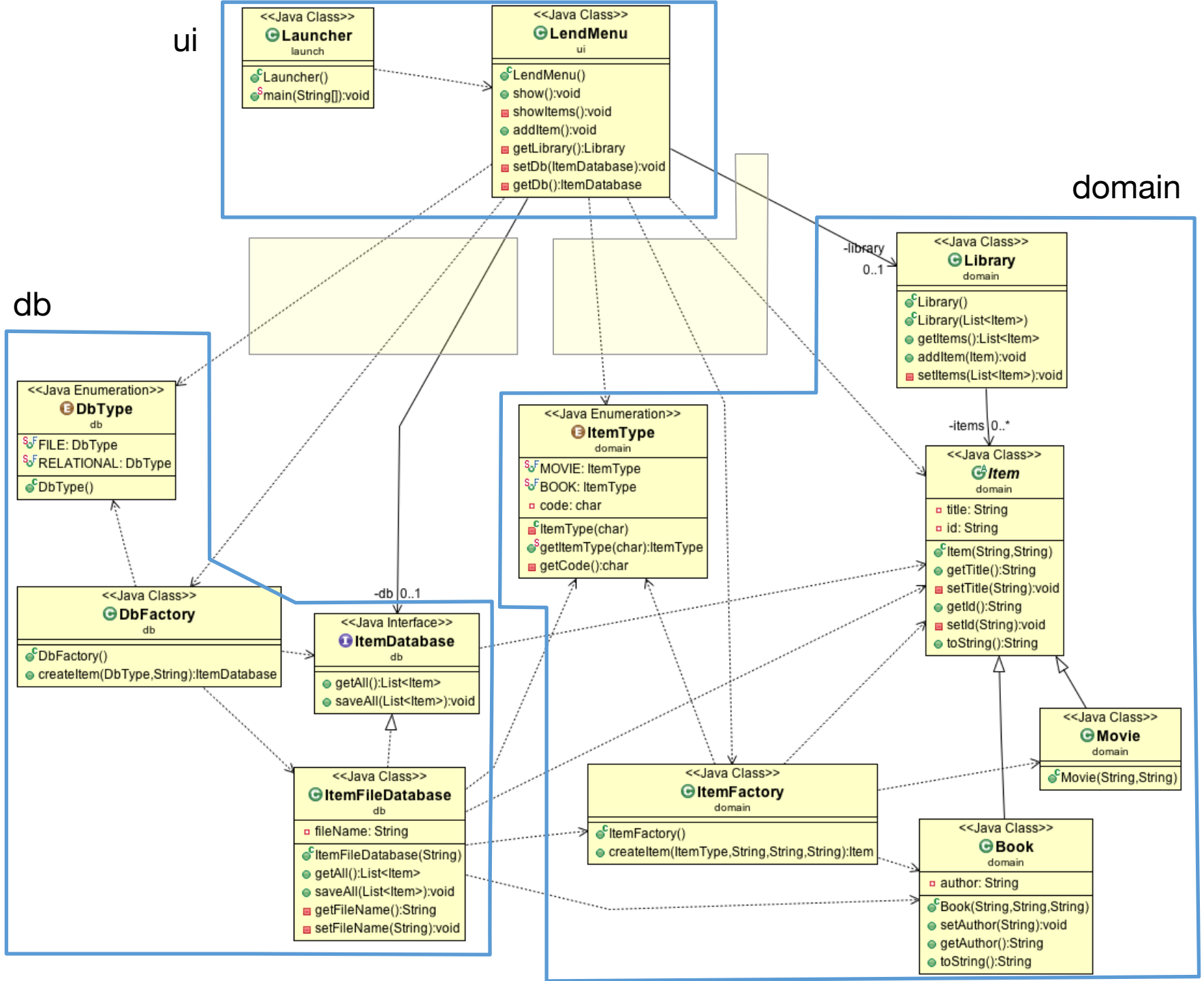
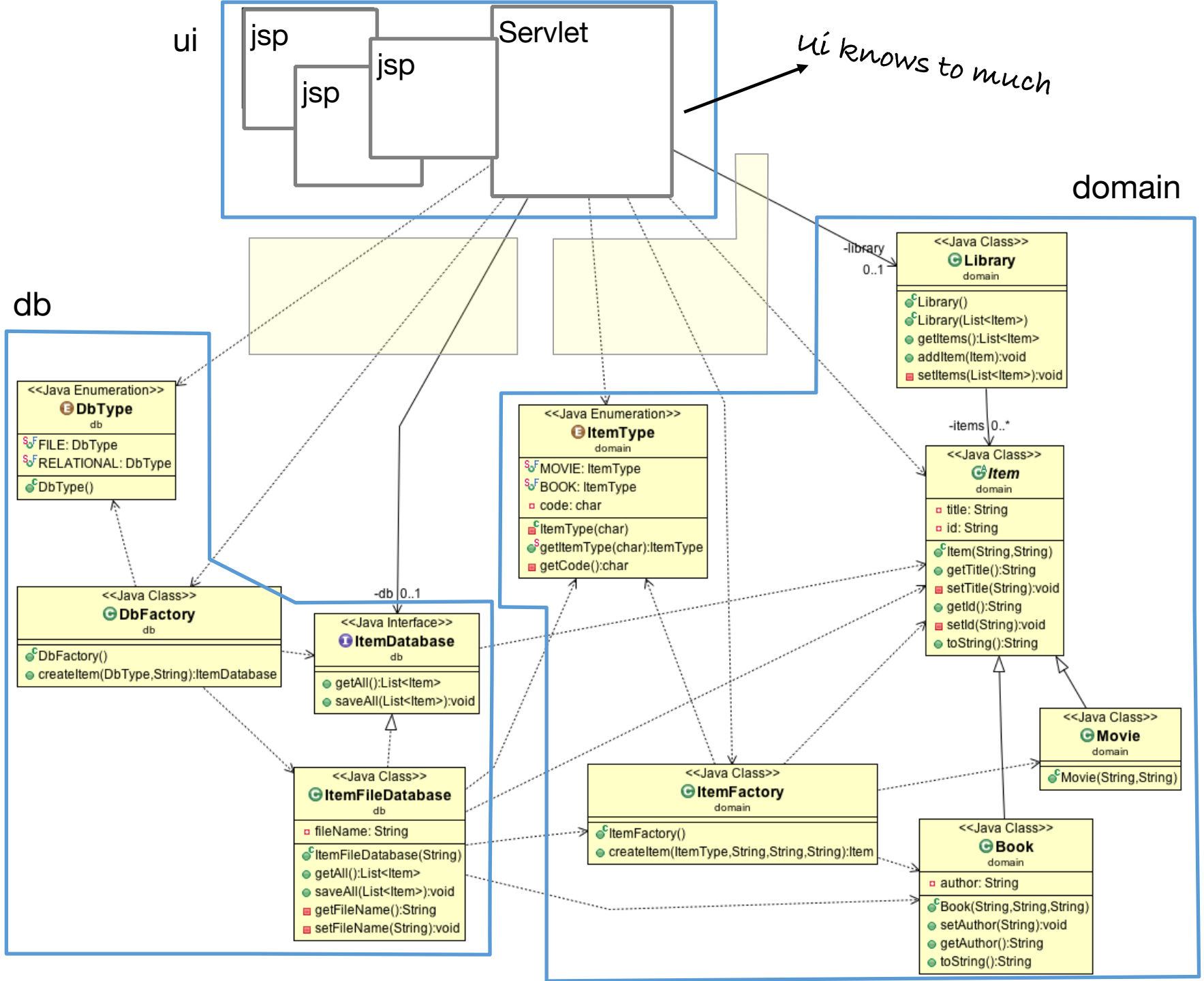


UC Leuven Limburg
MOVING MINDS



UC Leuven Limburg
MOVING MINDS

UC Leuven Limburg

MOVING MINDS

Facade

Facade: why?

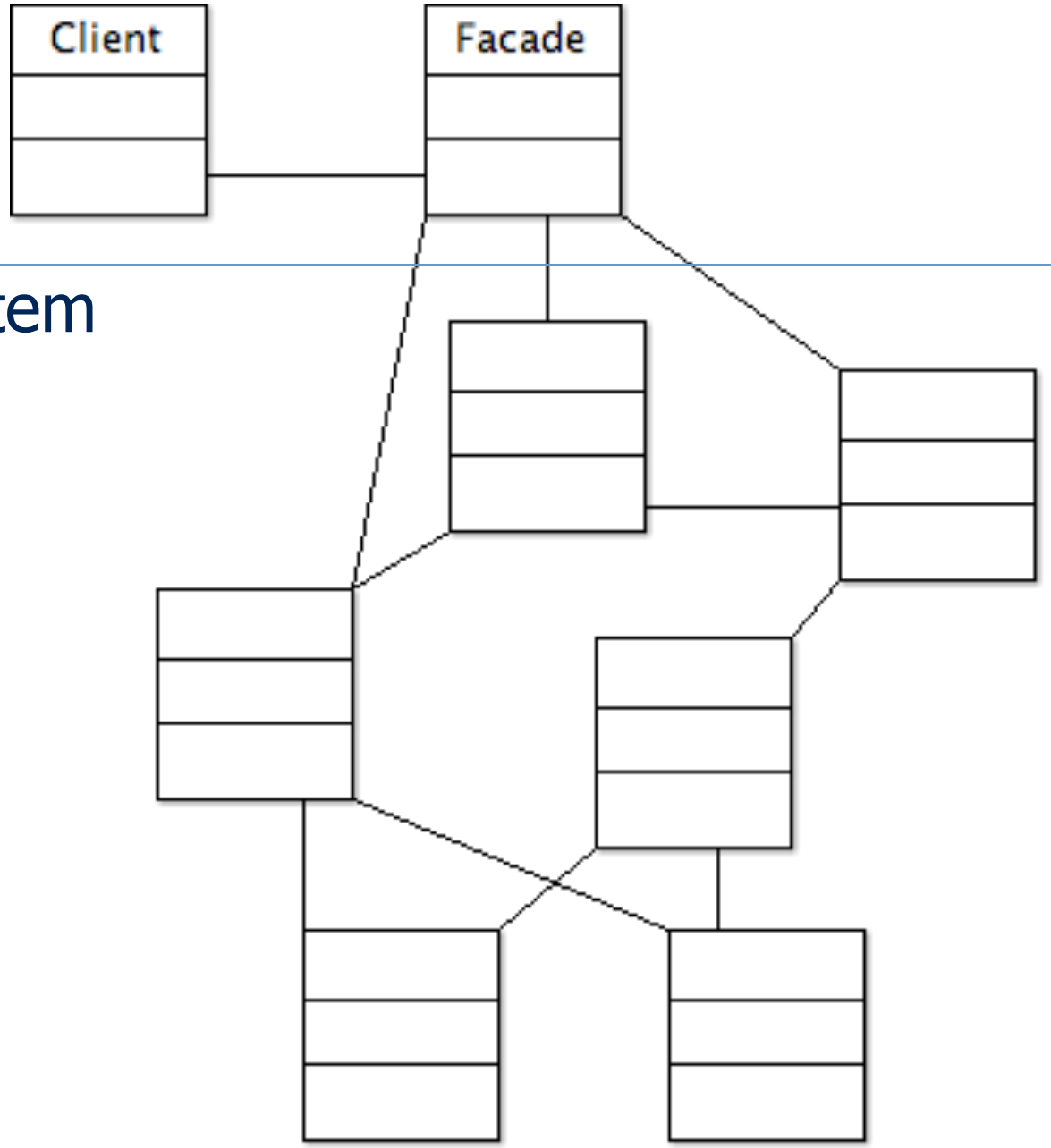
Simplified interface ...

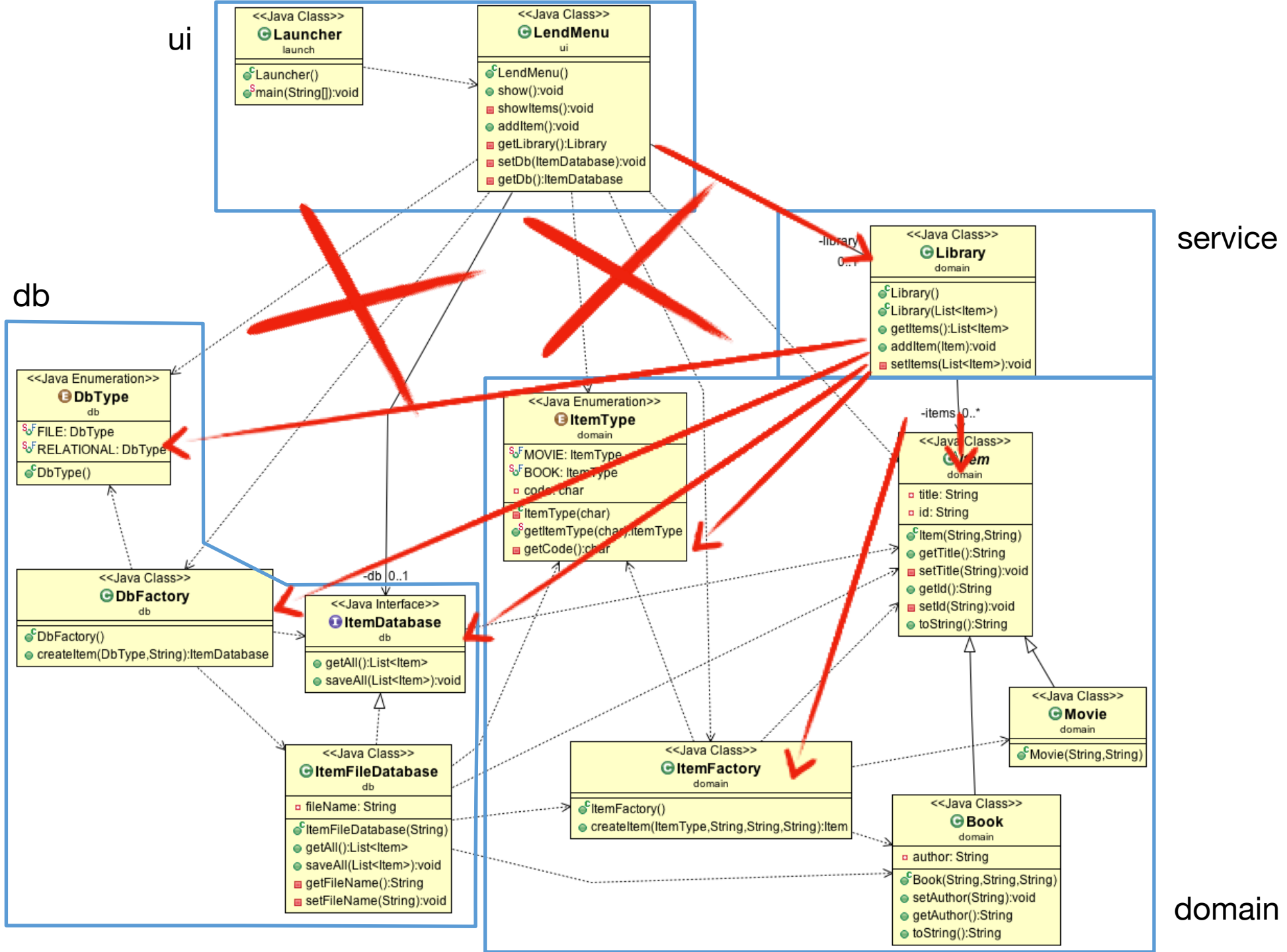
... representing a set of interfaces in a subsystem

How

- On top of all classes of subsystem: facade class
- Client talks to this class…
- … and does not need to know all the other classes

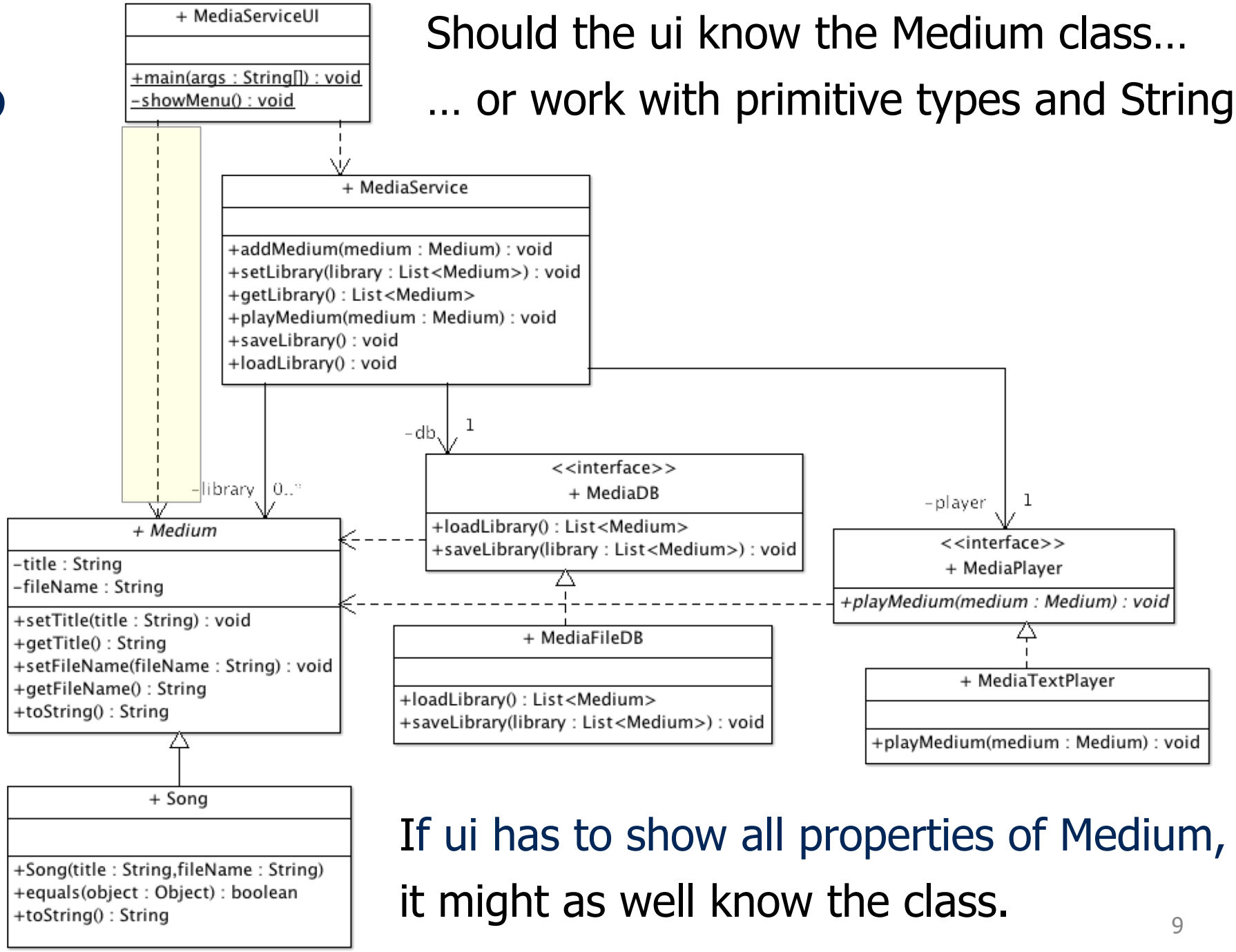
Classes subsystem





UC Leuven Limburg
MOVING MINDS

OK ?



Should the ui know the Medium class…

… or work with primitive types and Strings?

If ui has to show all properties of Medium, it might as well know the class.

Principle Of Least Knowledge

UC Leuven Limburg
MOVING MINDS

Principle Of Least Knowledge

```
public class SomeClass {
  private Subscription subscription ;
  ...
  public int calculateStudyEffort() {
    subscription.getCourse().getSubject().getCredit()
        //verdere code
  }
  ...
}
```

How many classes this code depends on? → **3**

Principle Of Least Knowledge

Guideline: only invoke methods that belong to:

- the object itself
- objects received as a parameter to the method
- any object the method creates or instantiates
- any components of the object

Facade: Summary

- Simplified interface ...
  - ... representing a set of interfaces in a subsystem
    - { object1.method1();
      object2.method1();
      object3.method1(); }

    → move to **facade.method();**

  - … for a method chain
    - { objectX.method1().method3().method2(); }

    → replace by **objectX.method();**

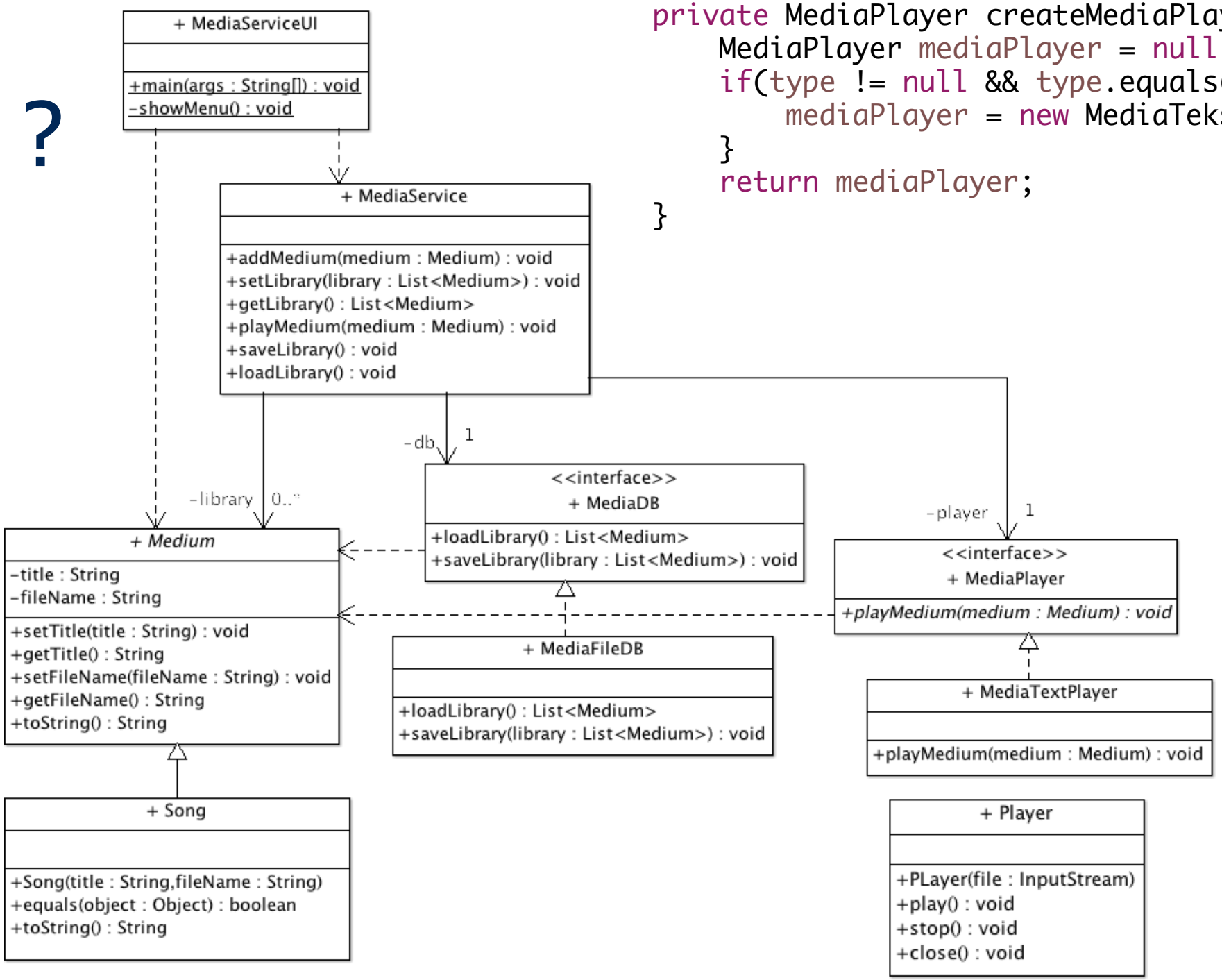
SOLID?

- SRP
  - OK: only intermediary
- OCP
  - NOK: new functionality → new methods
- LSP
  - Not applicable
- ISP
  - Not applicable
- DIP
  - Not applicable

?

UC Leuven Limburg
MOVING MINDS

?




```
private MediaPlayer createMediaPlayer(String type){
    MediaPlayer mediaPlayer = null;
    if(type != null && type.equals("tekst")){
        mediaPlayer = new MediaTekstPlayer();
    }
    return mediaPlayer;
}
```

Adapter

Design patterns

UC Leuven Limburg

MOVING MINDS

Adapter

Adapter: why?

- Adapt the interface of a class
- In order to enable classes with different (incompatible) interfaces to work together

Adapter: how?

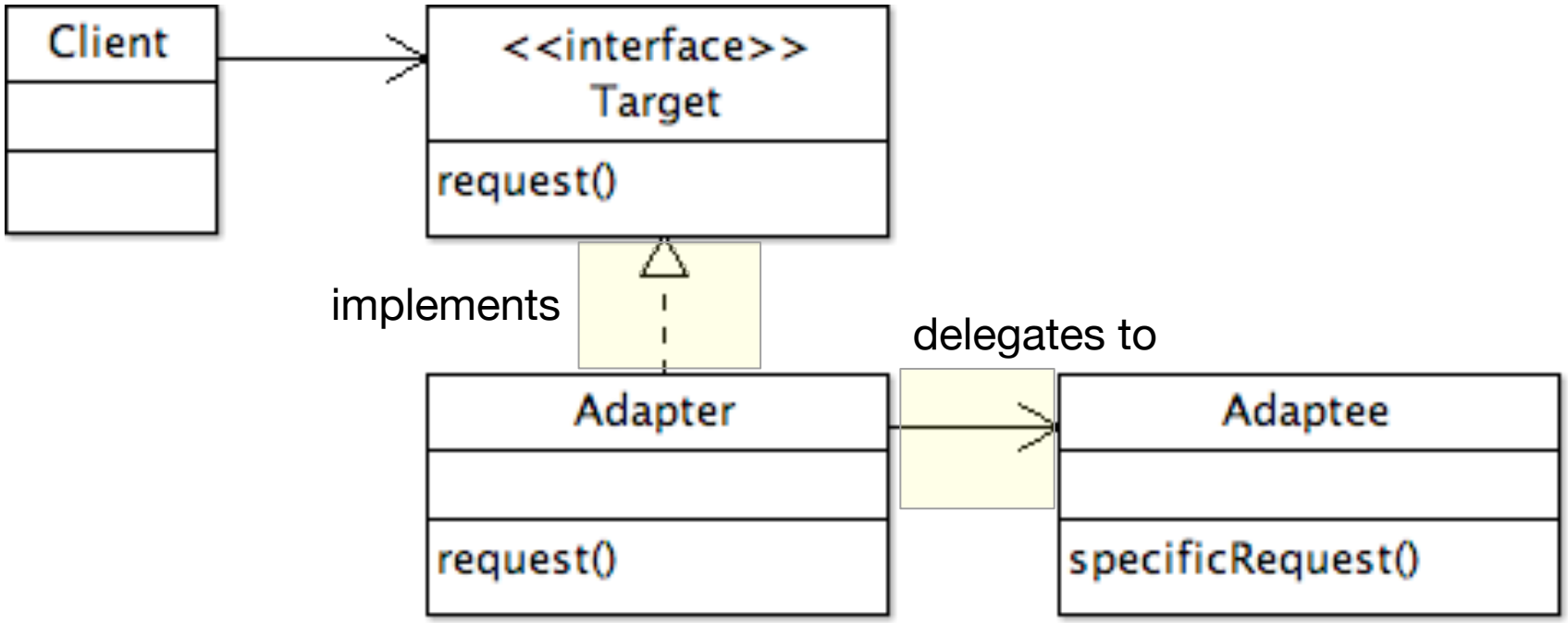
Adaptee

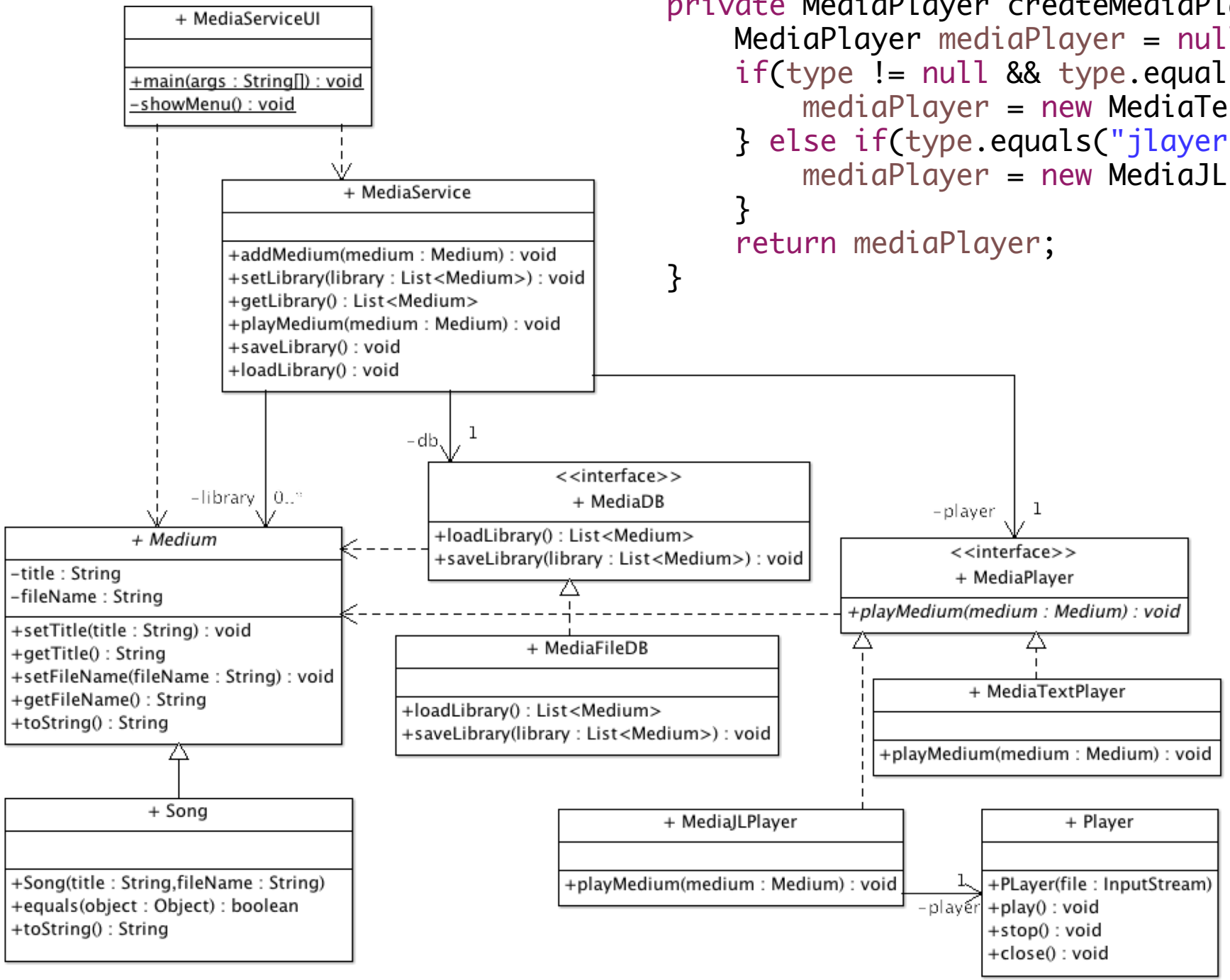
- has functionality needed by the client
- but does not implement expected (target) interface

Adapter

- implements target interface
- has an instance of adaptee
- does not execute request himself, delegates to adaptee

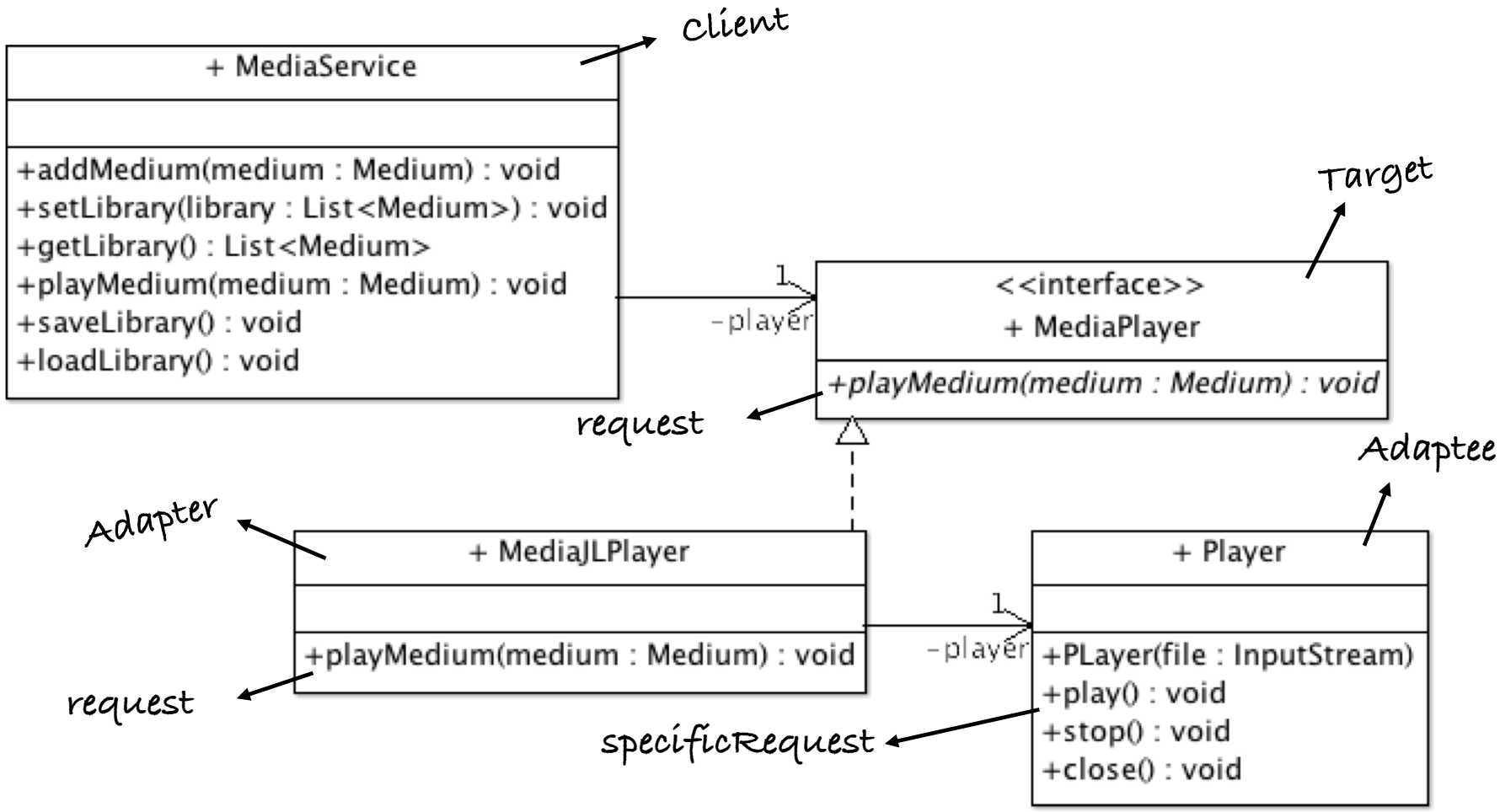
Adapter: class diagram






```
private MediaPlayer createMediaPlayer(String type){
    MediaPlayer mediaPlayer = null;
    if(type != null && type.equals("tekst")){
        mediaPlayer = new MediaTekstPlayer();
    } else if(type.equals("jlayer")){
        mediaPlayer = new MediaJLPlayer();
    }
    return mediaPlayer;
}
```

Class diagram MediaPlayer



Adapter vs. Facade

	Facade	Adapter
Are there preexisting classes?	Yes	Yes
Is there an interface we must design to?	No	Yes
Does an object need to behave polymorphically?	No	Probably
Is a simpler interface needed?	Yes	No

Adapter vs. Facade

- =
  - wrap existing class(es)

- **Adapter**
  - adapt existing interface
  - does (not) add new functionality

- **Facade**
  - simplify existing interfaces
  - does not add new functionality

SOLID?

- SRP
  - OK: adapter adapts
- OCP
  - OK
- LSP
  - Not applicable
- ISP
  - Not applicable
- DIP
  - OK

?