Software Engineering Essentials

UML Class Diagram - Object Design

Bernd Bruegge, Stephan Krusche, Andreas Seitz, Jan Knobloch
Chair for Applied Software Engineering — Faculty of Informatics

UML diagrams covered in this course

UML diagram
Structure diagram
Behavior diagram
Class diagram
Component diagram
Object diagram
Activity diagram
Interaction diagram
Use case diagram
State machine diagram
Deployment diagram
…
Package diagram
Sequence diagram
…
Communication diagram

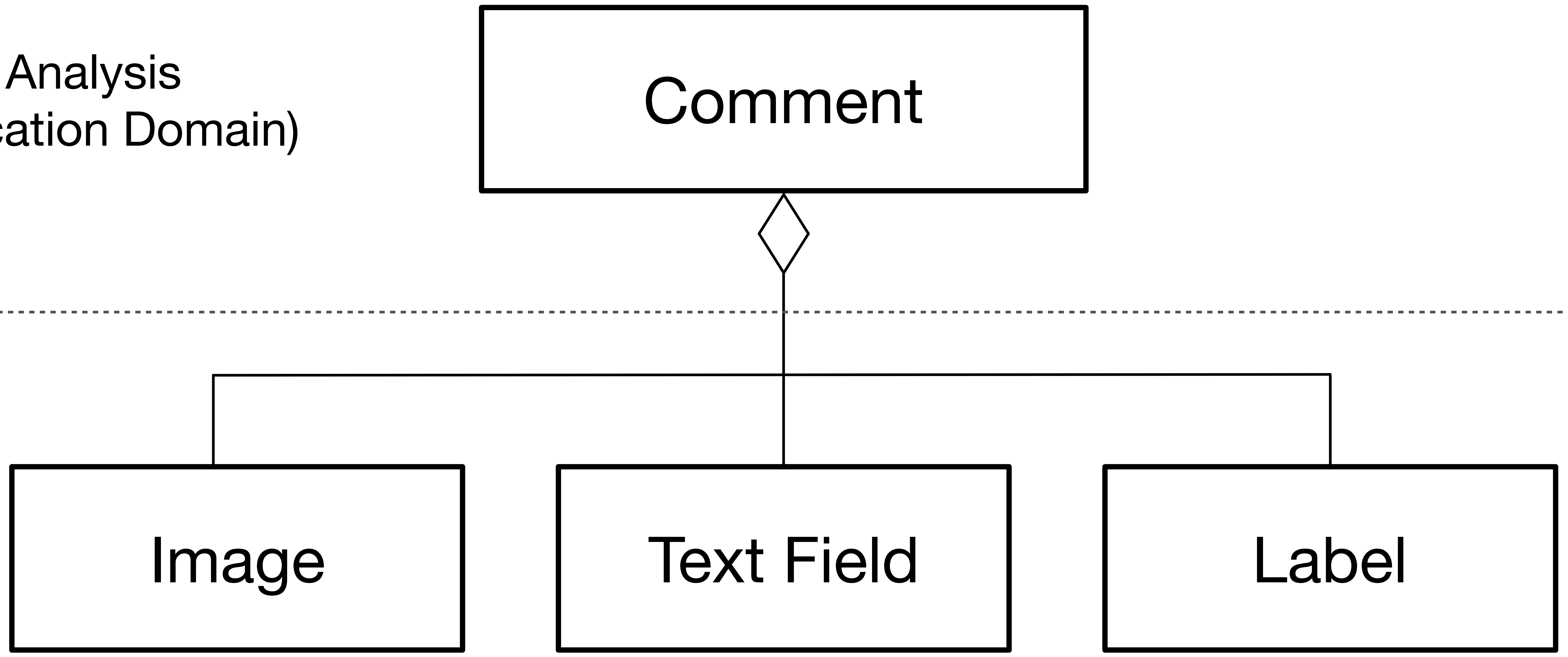
Activities to enrich class diagrams during object design

1) Add solution domain specific classes, attributes and methods

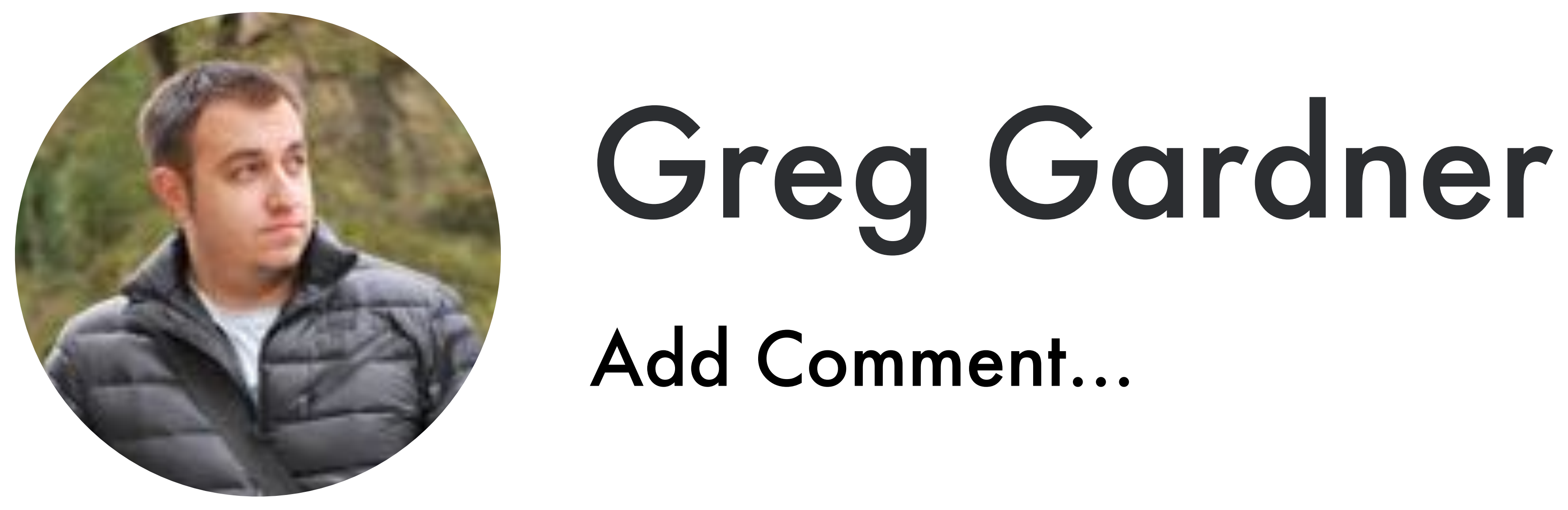
2) Specify interfaces: signatures and visibility of attributes and methods

Adding Solution Domain specific Classes

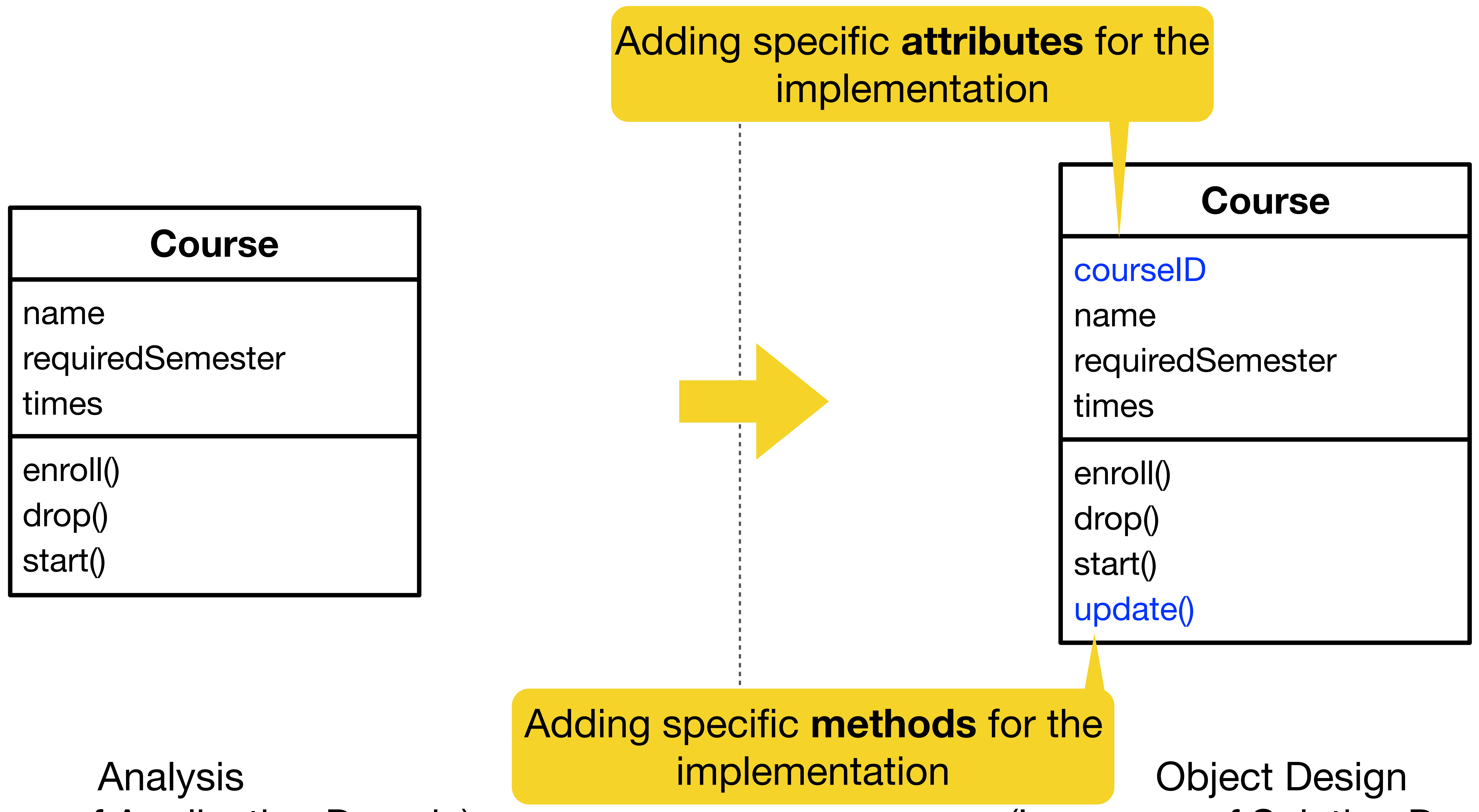
Requirements Analysis
(Language of Application Domain)



Object Design
(Language of Solution Domain)



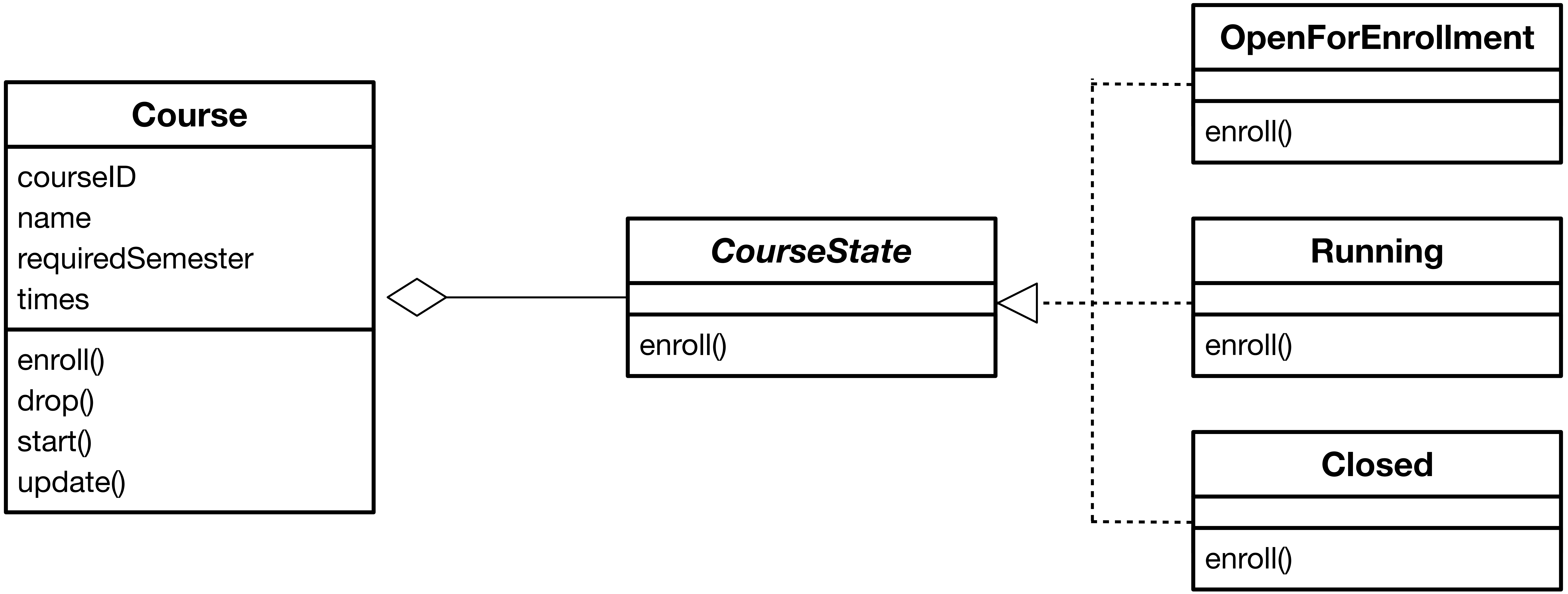
Adding Solution Domain specific Attributes & Methods



Analysis
(Language of Application Domain)

Object Design
(Language of Solution Domain)

Example: Applying the State Pattern



Specifying Interfaces

Interface specification during **requirements analysis**

We covered this in unit
Class Diagram

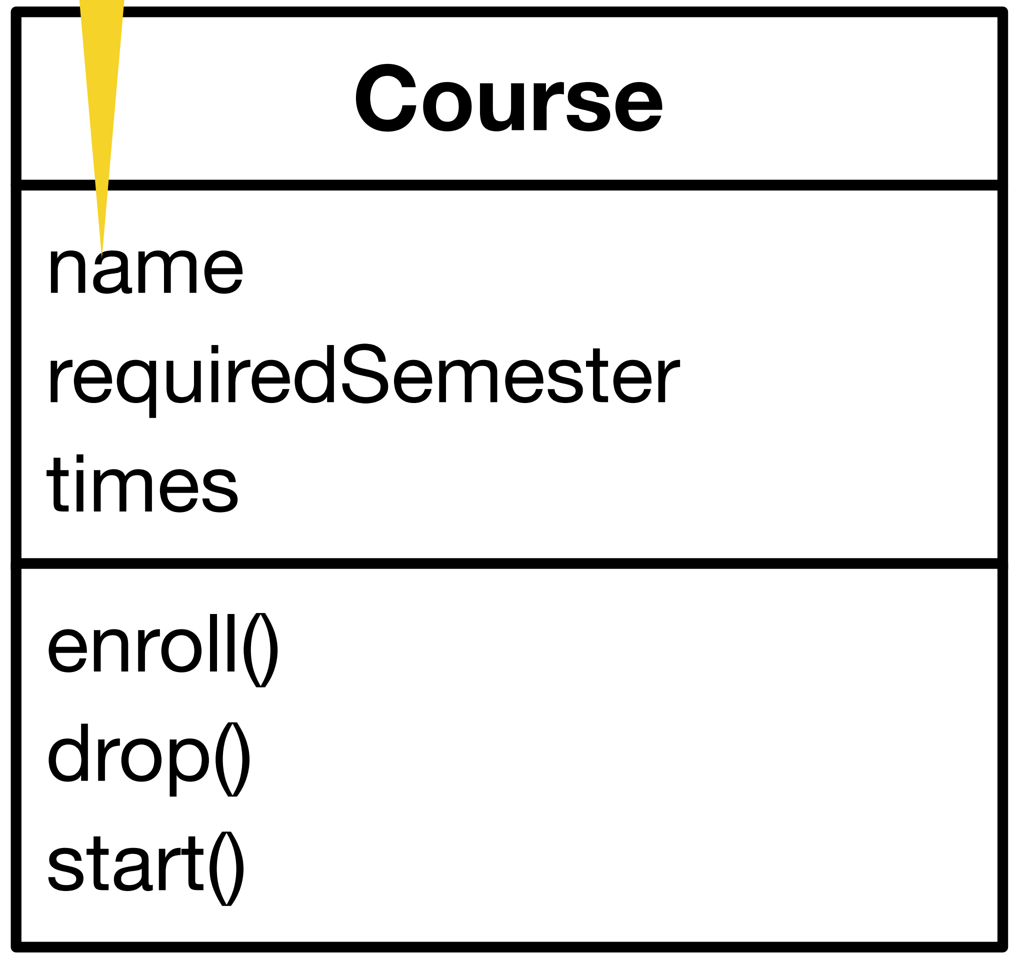
- Identification of attributes and operations
- No need to specify types or their parameters

Interface specification during **object design**

- Add type signature information
- Add visibility information

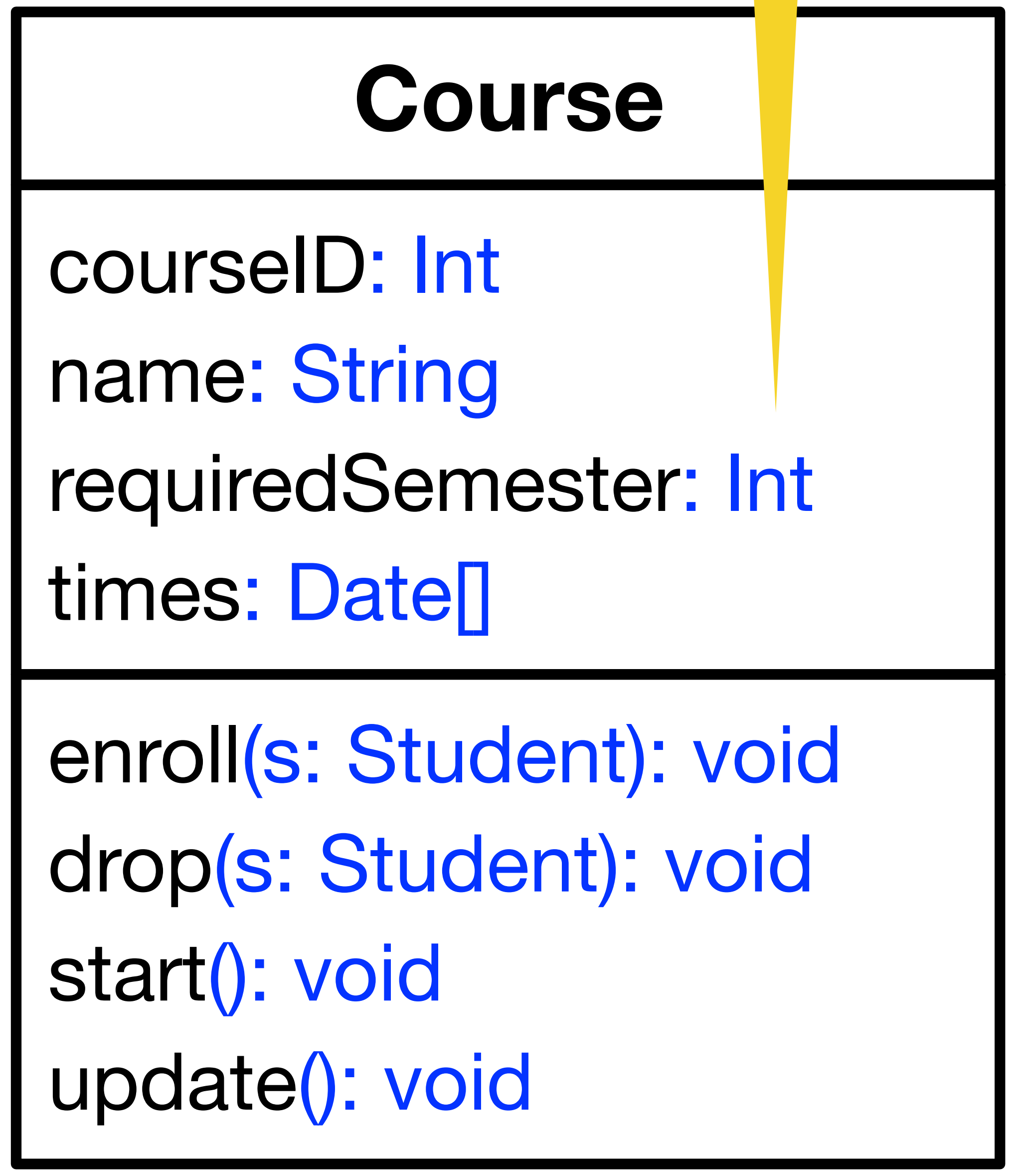
Adding Type Signature Information

During **analysis**: attributes and methods with type information are ok but not required



Analysis
(Language of Application Domain)

During **object design**: we must specify the signature for each method, the types for all attributes



Object Design
(Language of Solution Domain)

Adding Visibility Information in UML

Class user ("Public"): +

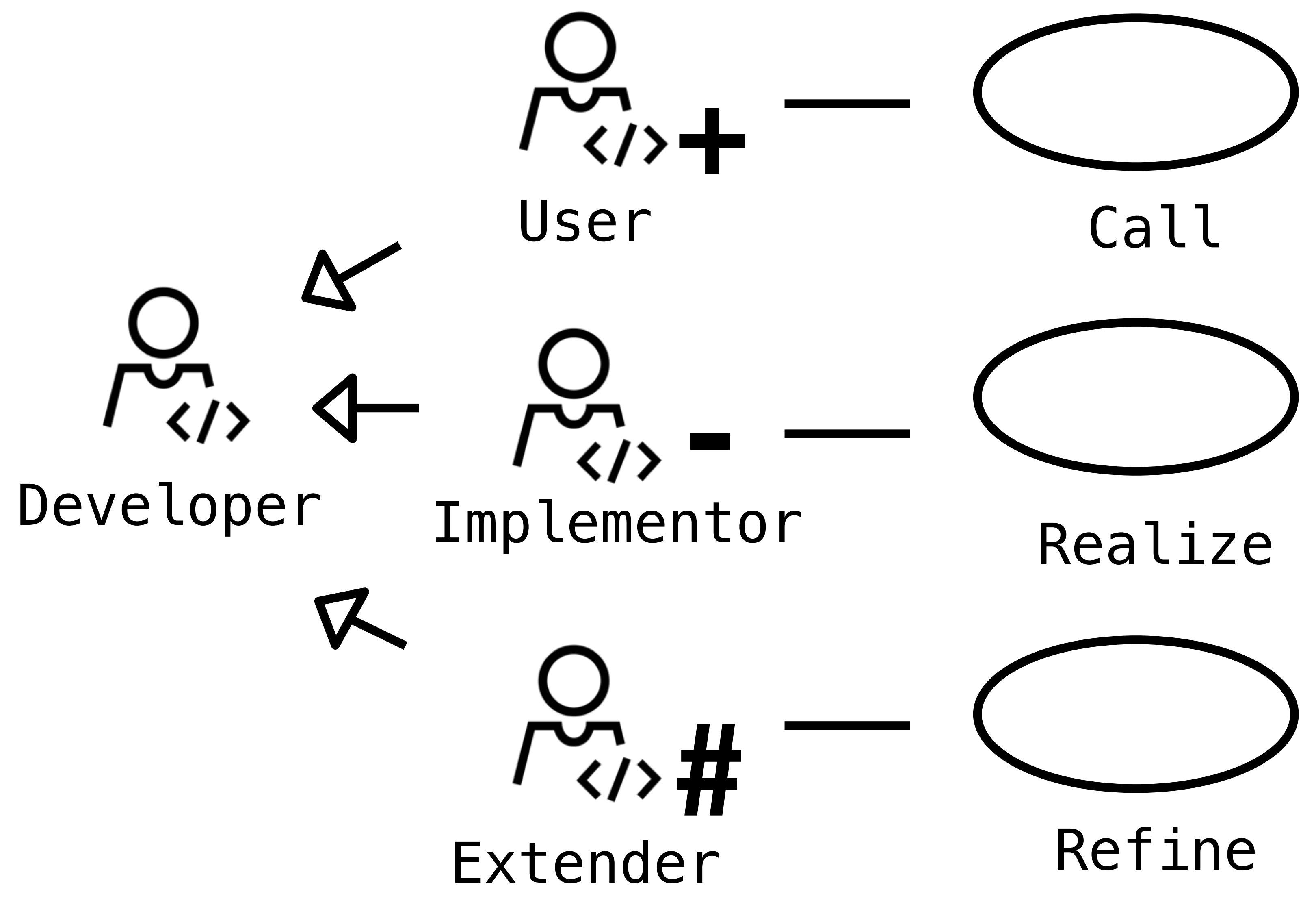
- Public attributes/methods can be accessed by any class

Class implementor ("Private"): -

- Private attributes/methods can be accessed within a class

Class extender ("Protected"): #

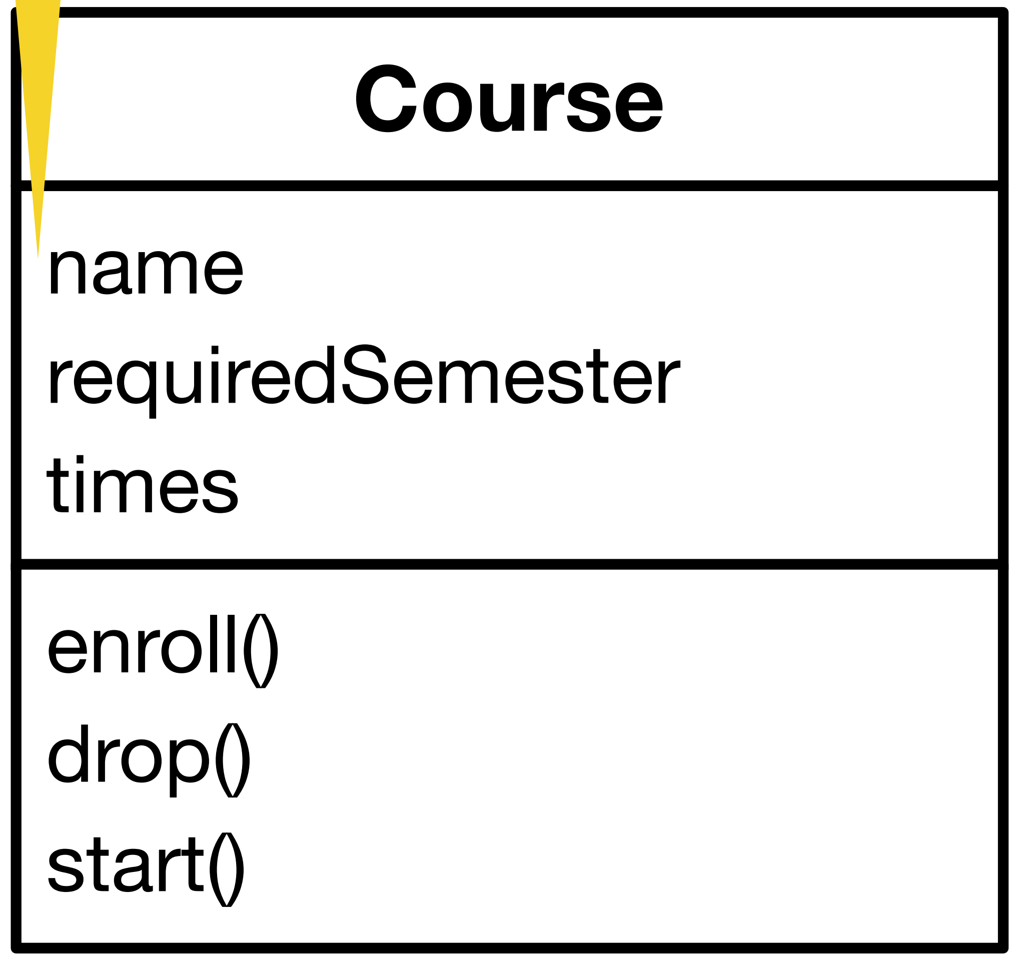
- Protected attributes/methods can be accessed by the class in which they are defined and by any descendent of the class



UML visibilities similar to Java visibilities except for packaging rules.

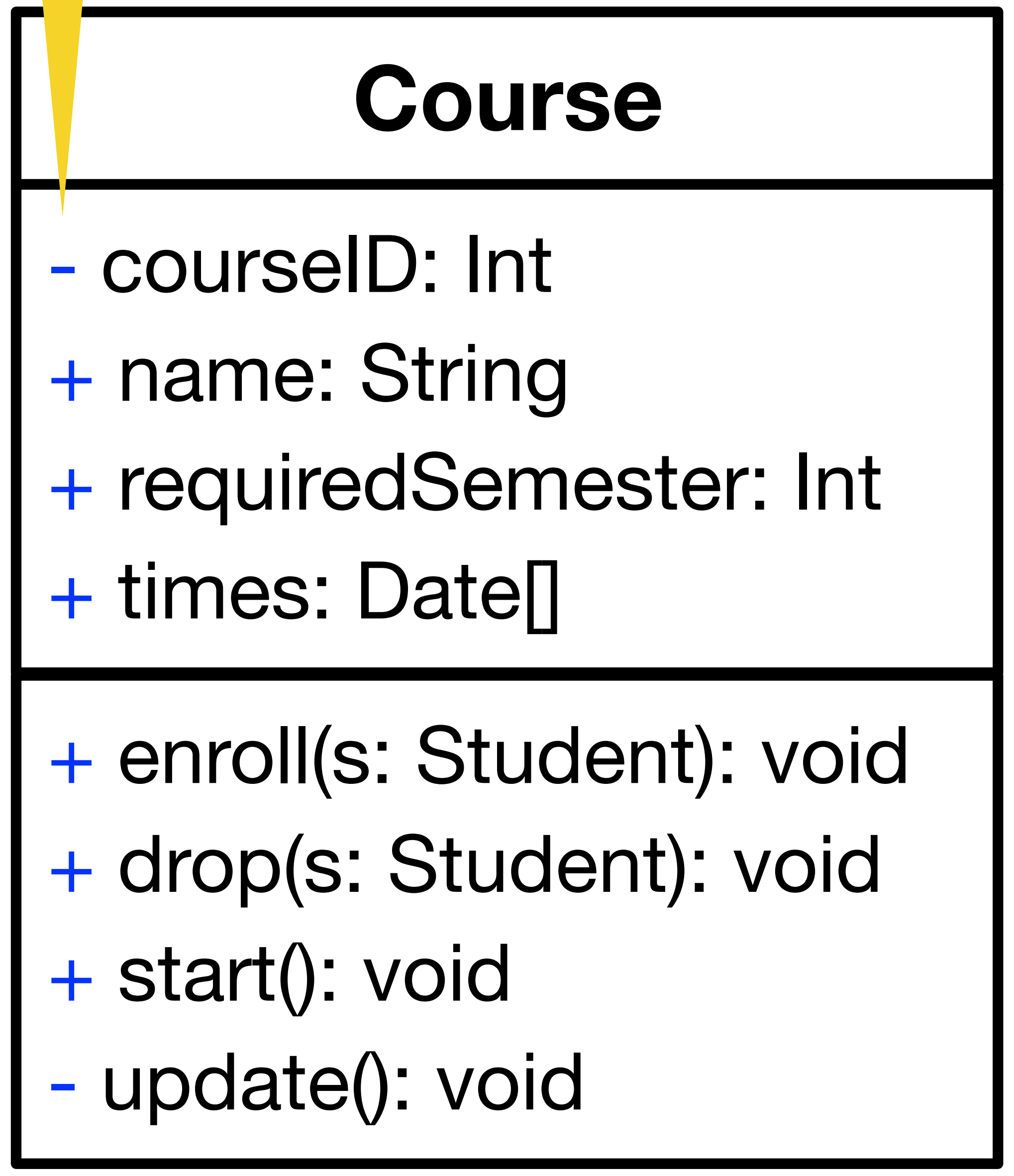
Adding Visibility Information

During **analysis**: attributes and methods without visibility information



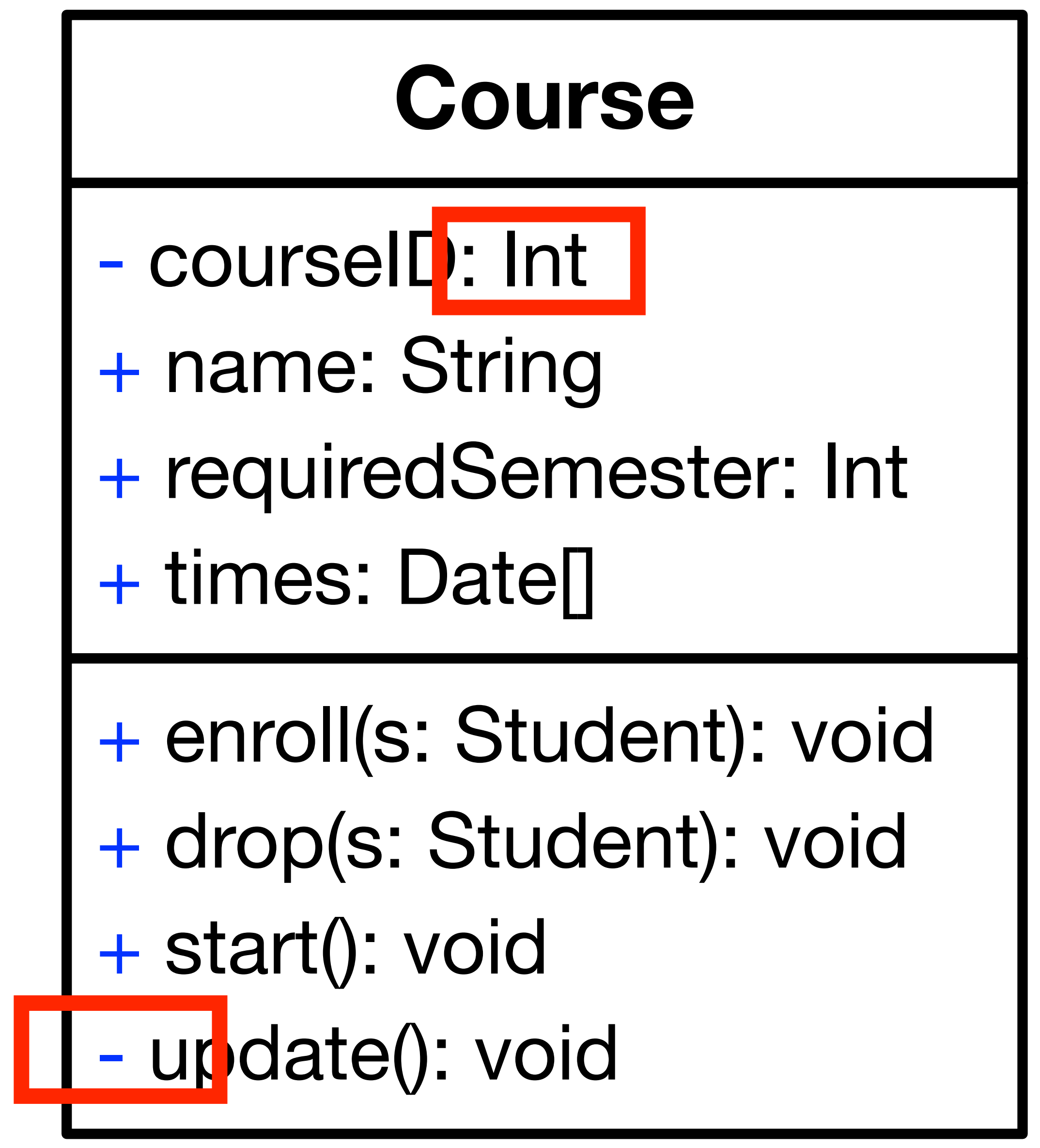
Analysis
(Language of Application Domain)

During **object design**: we must specify the visibility for each attribute and method



Object Design
(Language of Solution Domain)

Implementation of UML Visibility in Java



Object Design

```
public class Course {

  private Integer courseId;
  public String name;
  public Integer requiredSemester;
  public Date[] times;

  public void enroll(Student s) {…}
  public void drop(Student s) {…}
  public void start() {…}
  private void update() {…}

}
```

type information

signature information

visibility information

Implementation

Information Hiding Heuristics

- Carefully define the public interface for classes as well as subsystems
    - For subsystems use a façade design pattern if possible
- Always apply the “Need to know” principle:
    - Only if somebody needs to access the information, make it publicly possible
- The fewer details a class user has to know
    - The easier the class can be changed
    - The less likely the class user will be affected by any changes in the class implementation
- Good rule: Make attributes always private

Software Engineering Essentials

UML Class Diagram - Object Design

Bernd Bruegge, Stephan Krusche, Andreas Seitz, Jan Knobloch
Chair for Applied Software Engineering — Faculty of Informatics