Responsibility Driven Design

Charlotte Pierce

COS20007 Object Oriented Programming

SWINBURNE UNIVERSITY OF TECHNOLOGY

Software development involves providing instructions for an unintelligent computer

Developers work in teams to build software solutions, which typically contain millions of instructions

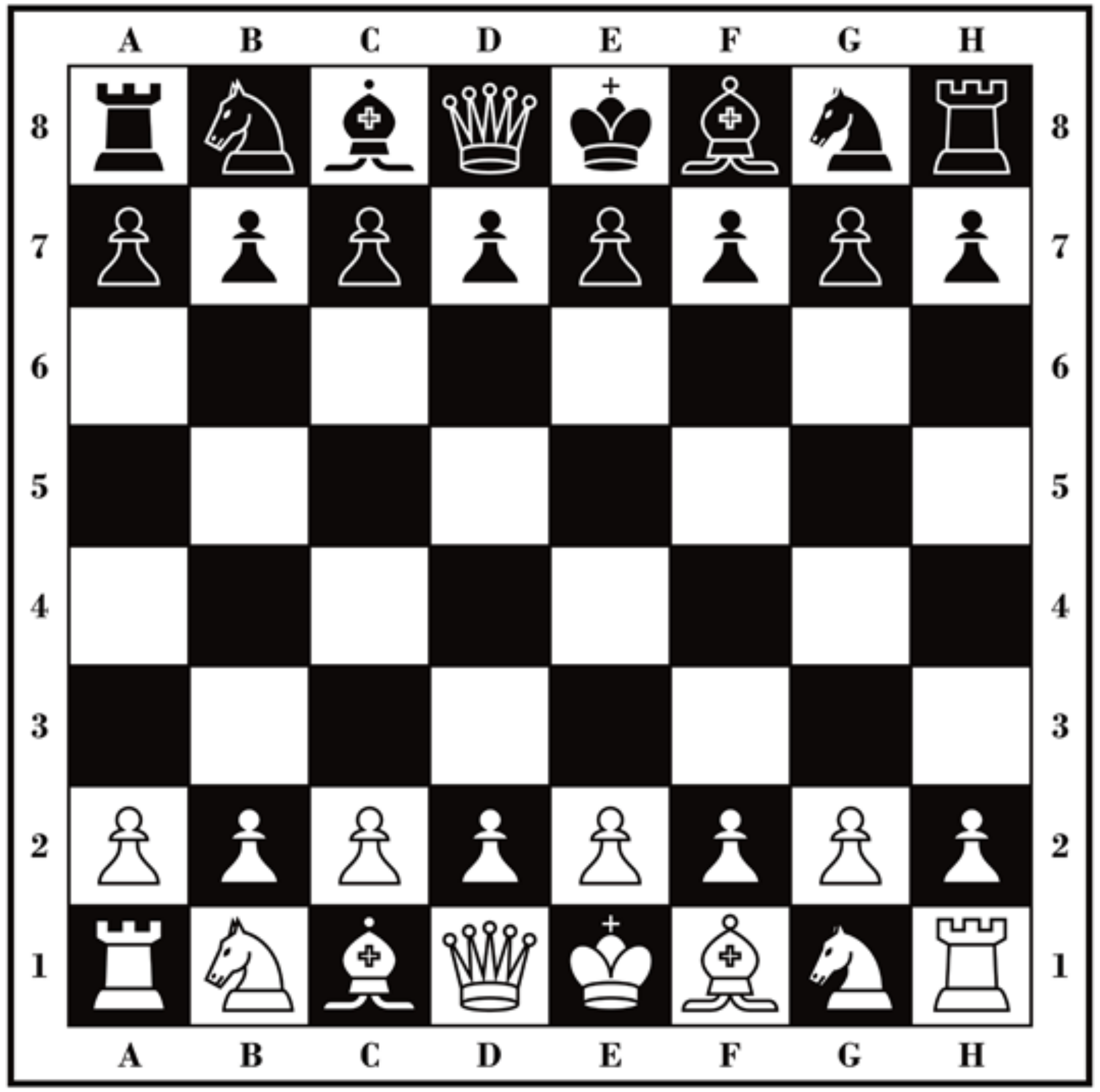
Seeing how a solution will work requires clear communication

Effective software design includes picturing the solution and having a common understanding

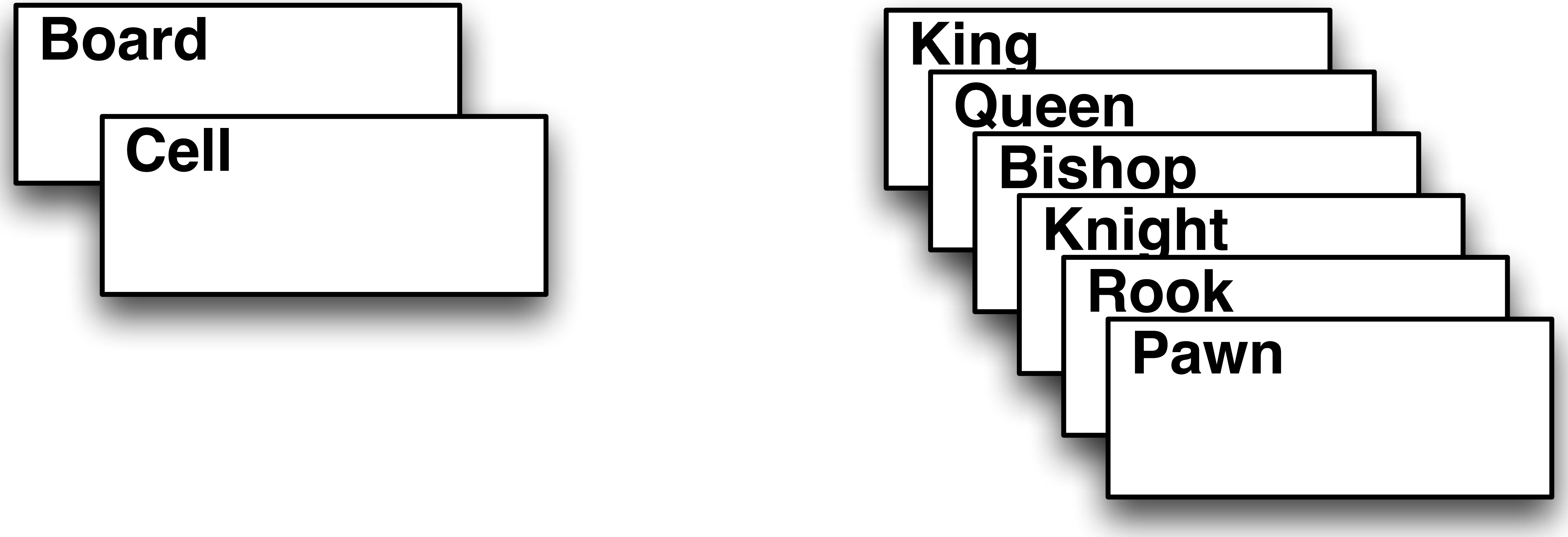
Create effective OO designs using Roles, Responsibilities, and Collaborations

Step 1: Define the purpose for objects in your program using **Roles**

Picture the problem domain and identify **candidate** roles (nouns are a good start)

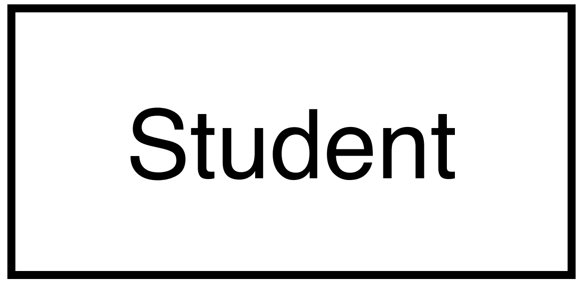


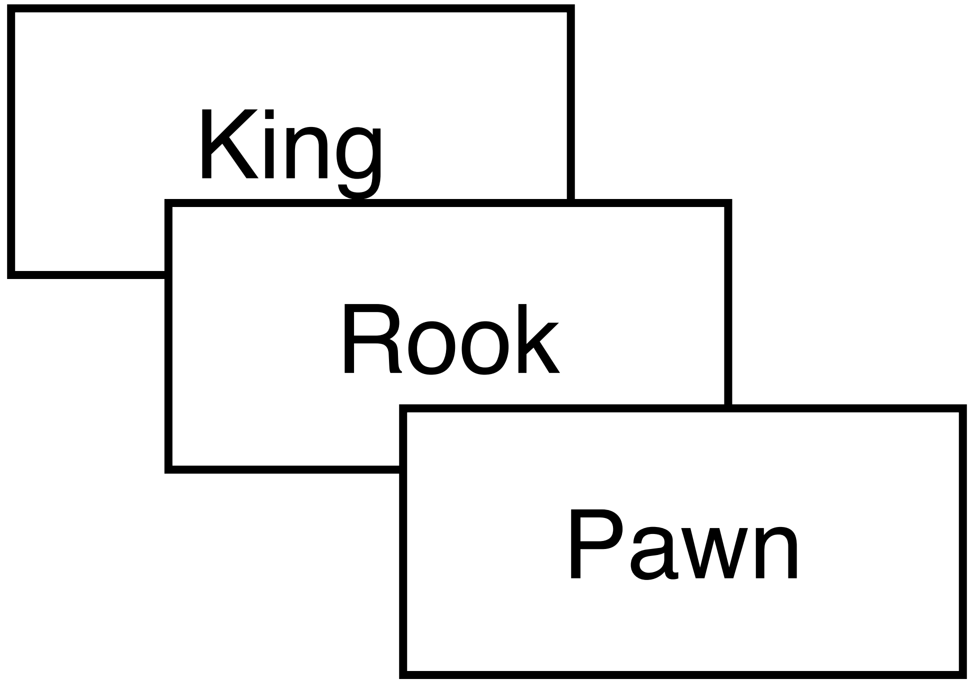
Explore candidate roles using CRC cards



CRC = candidate role, responsibility, collaborations

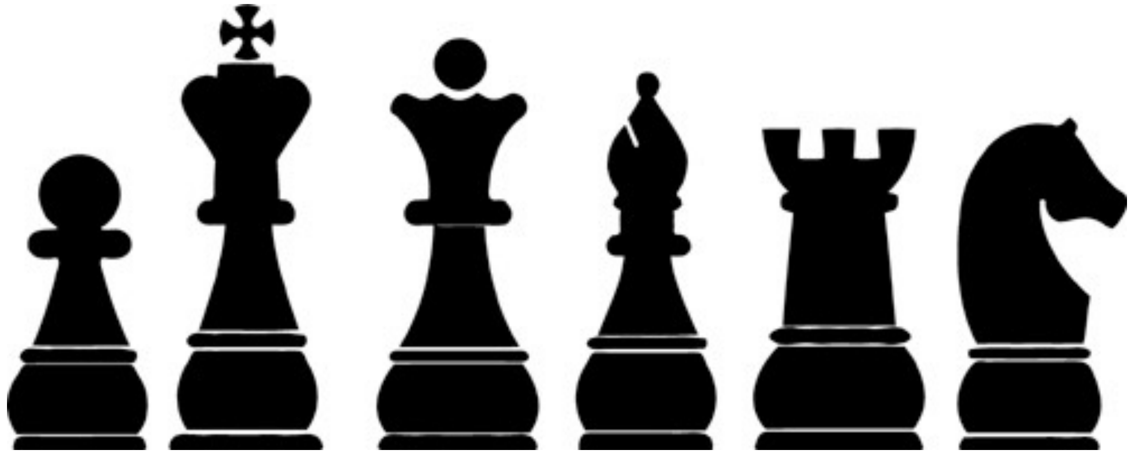
Draw boxes for classes in UML class diagrams to communicate static structure



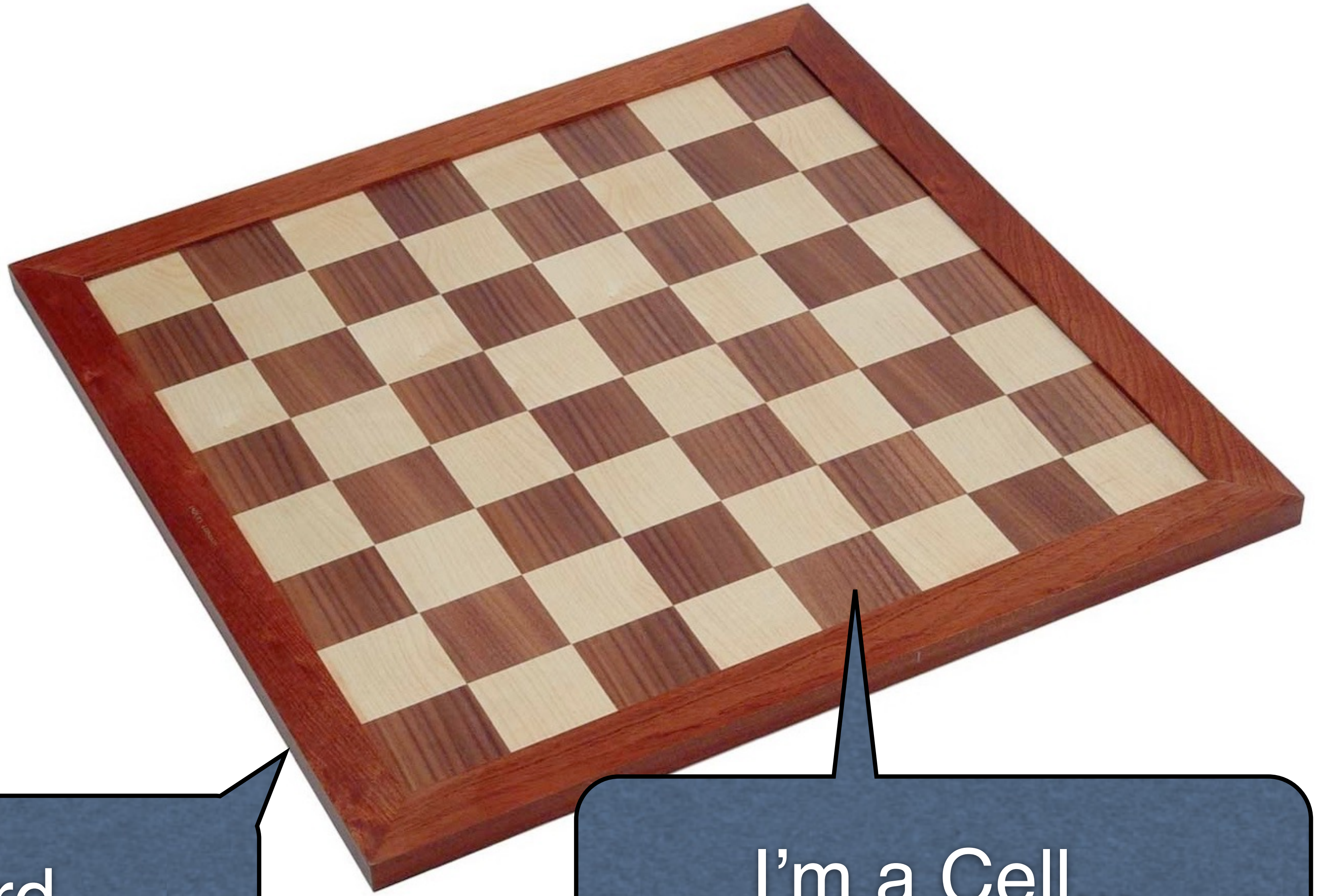


Step 2: Define
responsibilities
for each candidate role

Picture roles as having responsibilities within your overall solution



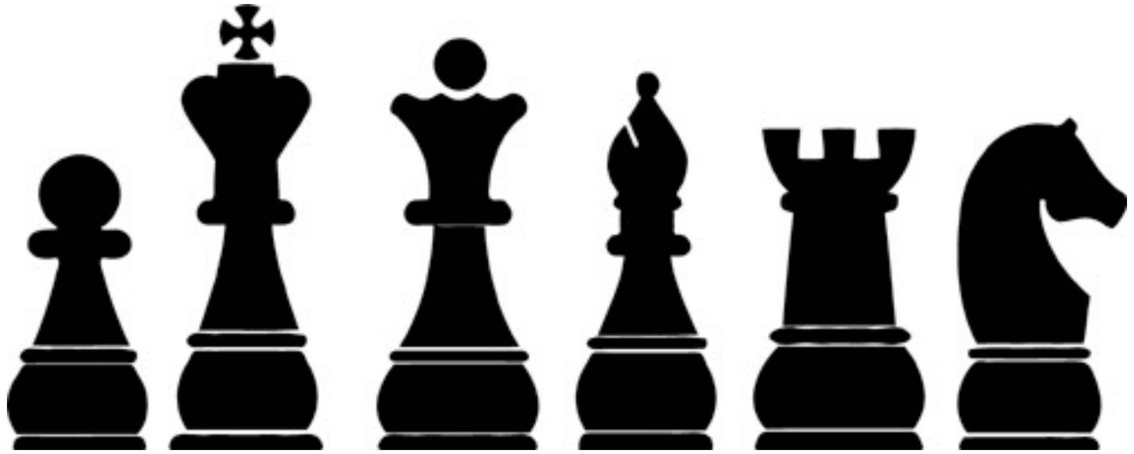
I’m a Bishop...
I’m responsible for...



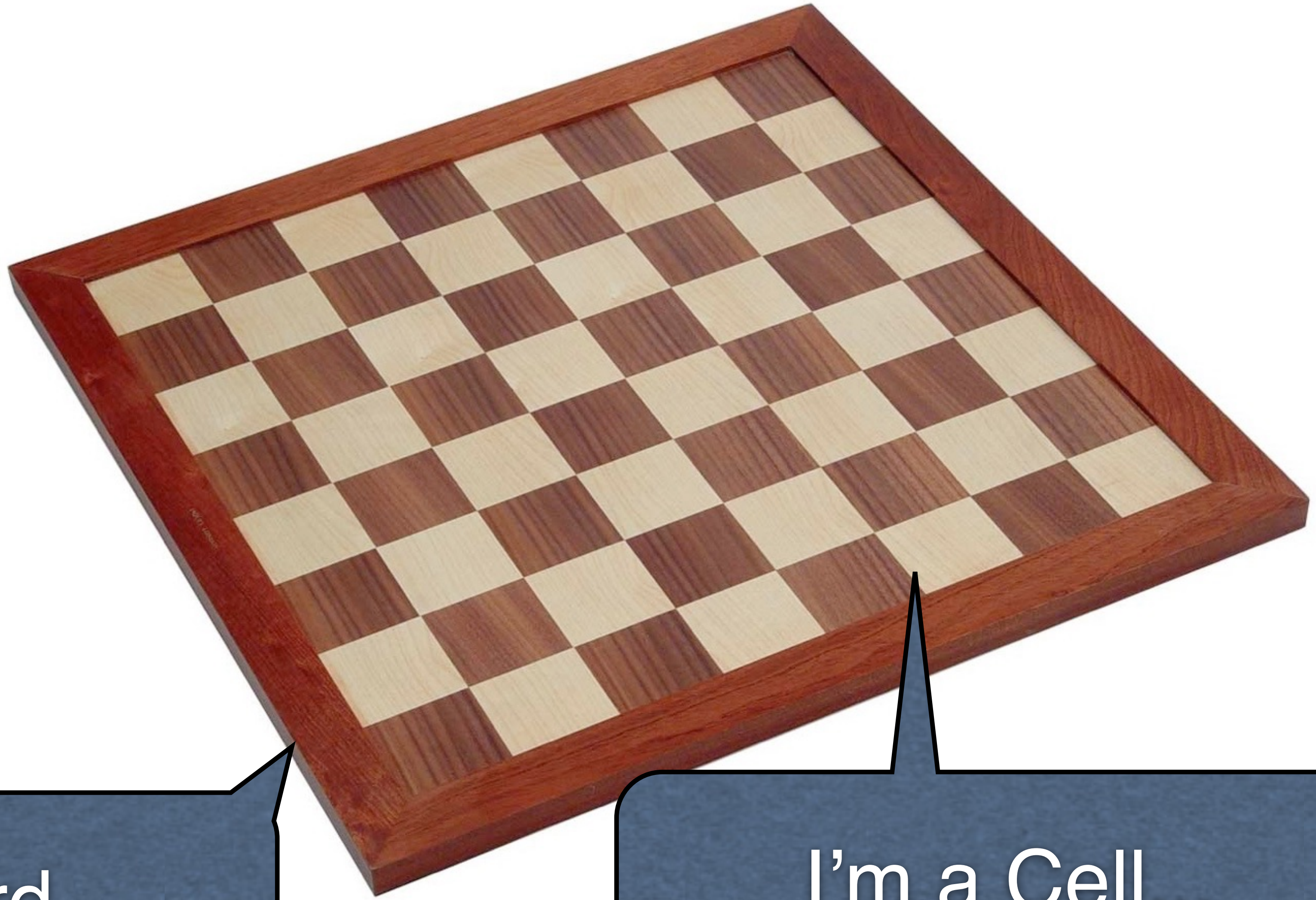
I’m a Board...
I’m responsible for...

I’m a Cell...
I’m responsible for...

Include responsibilities to **know** things, this forms the data for your program



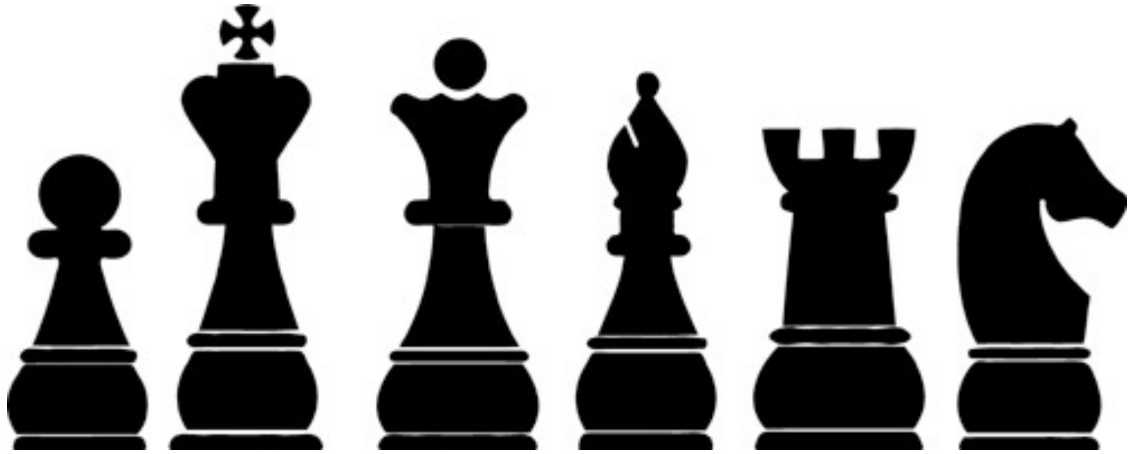
I’m a King...
I know my colour...



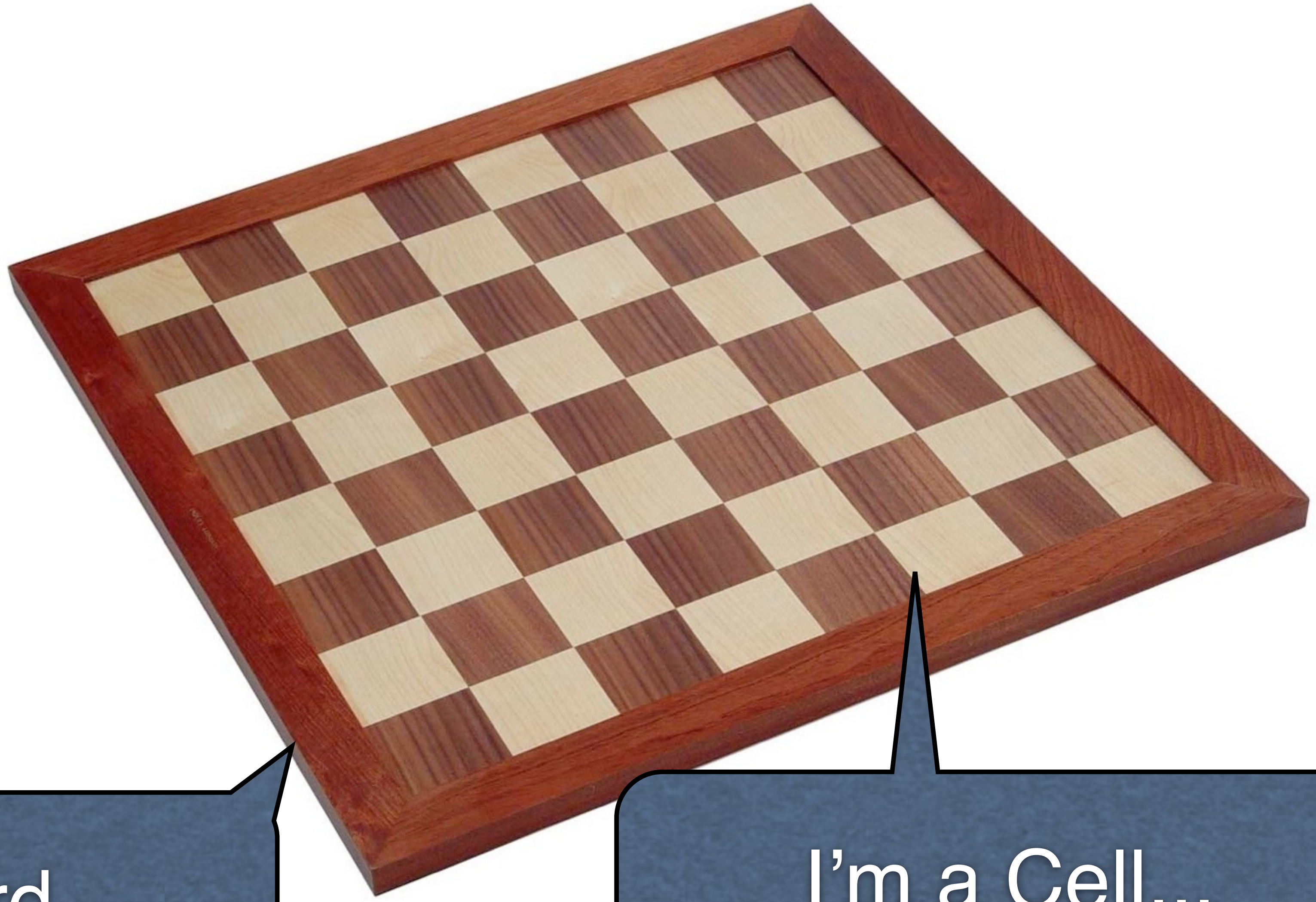
I’m a Board...
I know all of the

I’m a Cell...
I know my occupant.

Include responsibilities to **do** things, these become methods in the solution



I’m a Pawn...
I can be a Queen.



I’m a Board...
I can move pieces.

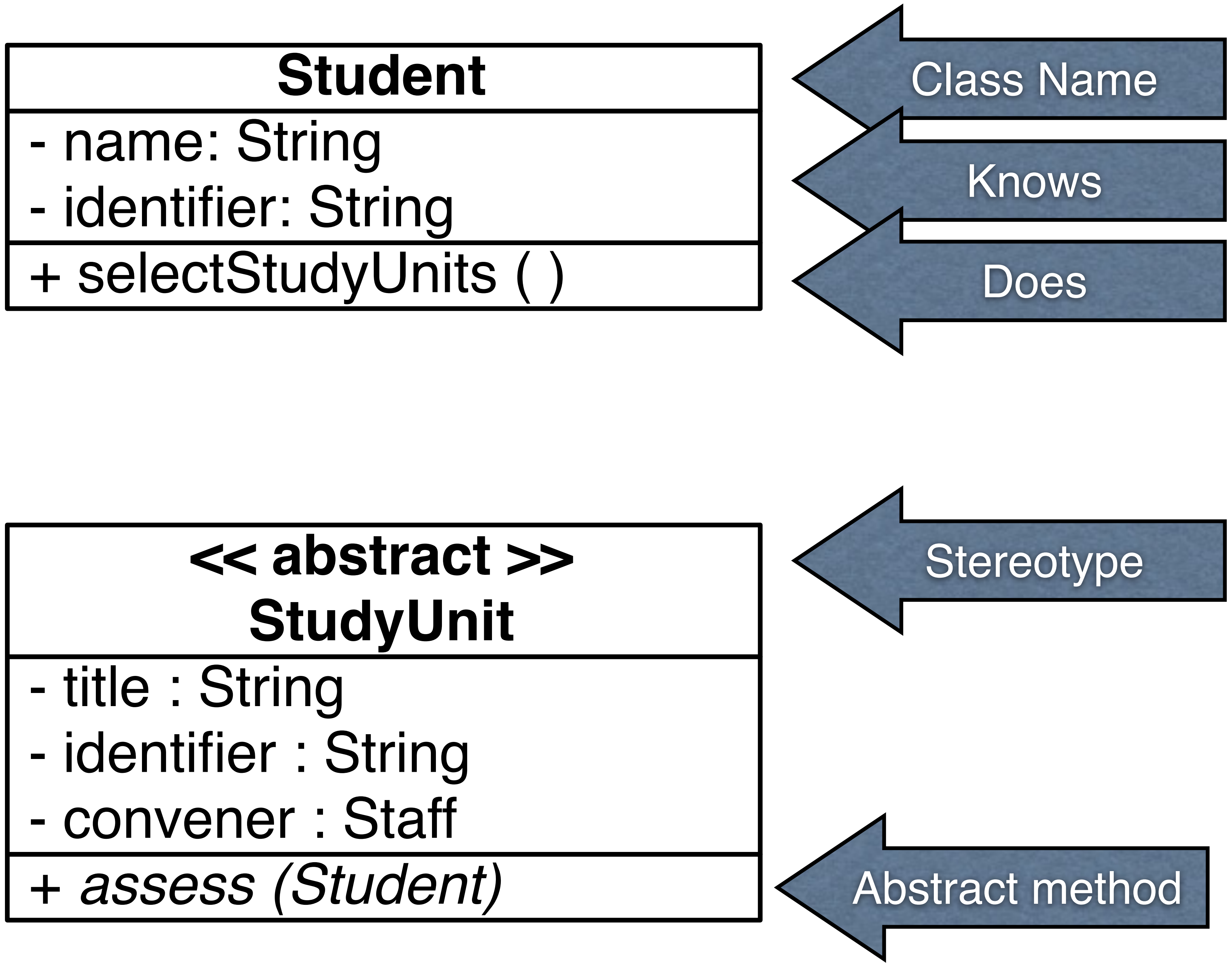
I’m a Cell...
I can hold a piece.

Explore responsibilities using CRC cards

Pawn

knows its color
knows its valid moves
can become a Queen
can take another piece

Document responsibilities in UML class diagrams



Student

- name: String
- identifier: String

+ selectStudyUnits ( )

Class Name

Knows

Does

<< abstract >>
StudyUnit

- title : String
- identifier : String
- convener : Staff

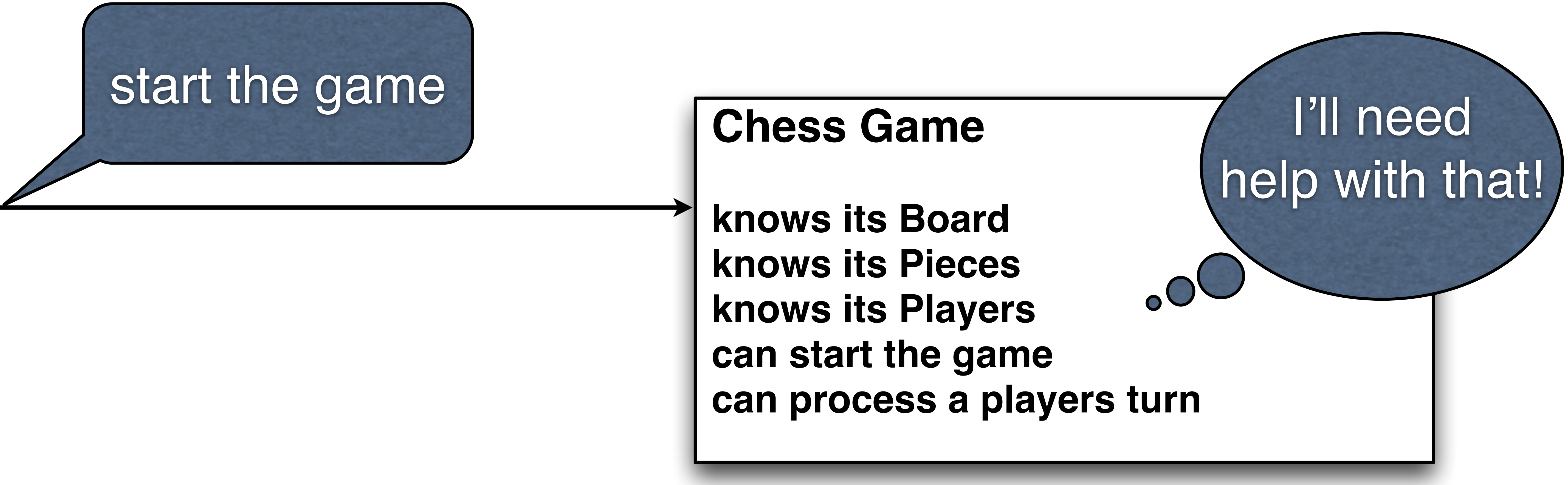
+ *assess (Student)*

Stereotype

Abstract method

Step 3: **Collaborate** with other objects to meet responsibilities

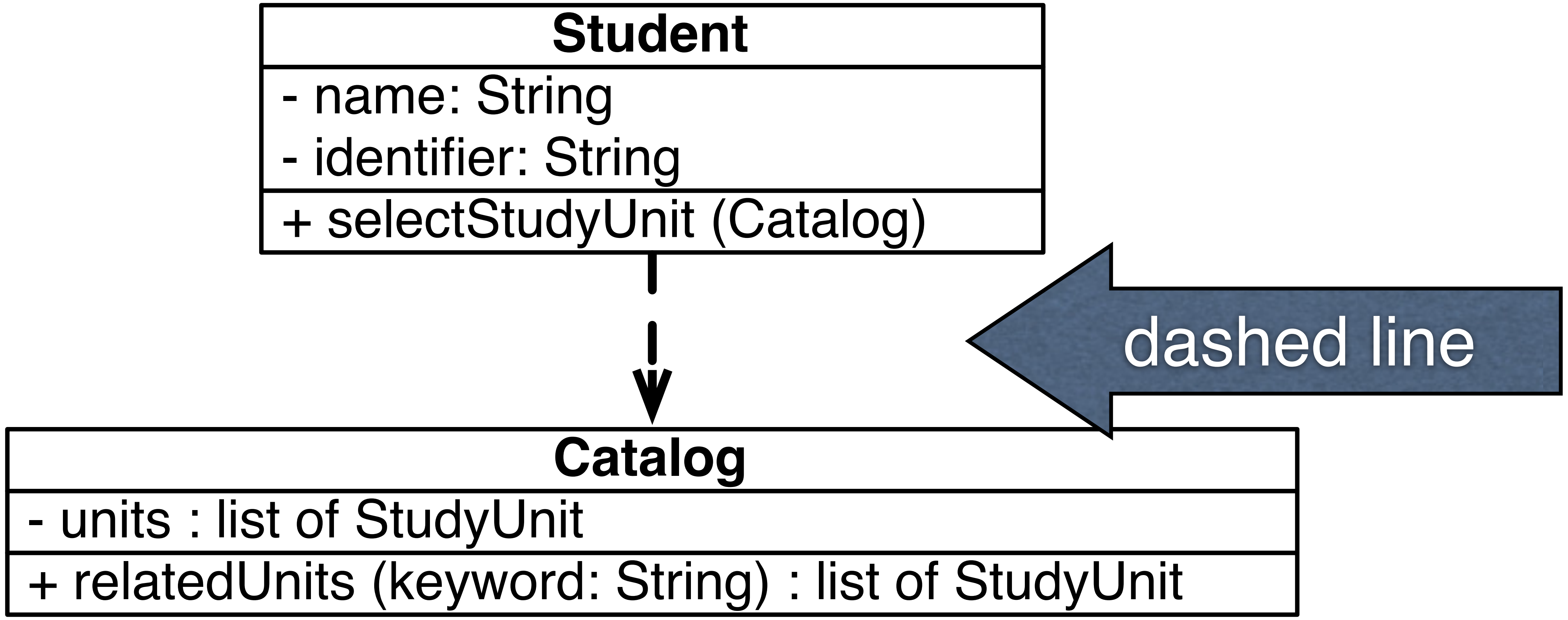
When asked to perform a task, objects can ask others for help



Think of collaborations as a **client/supplier** interaction or as a contract

Use the different kinds of relationships to help identify possible links

Dependence involves temporary use of another object



Student

- name: String

- identifier: String

+ selectStudyUnit (Catalog)

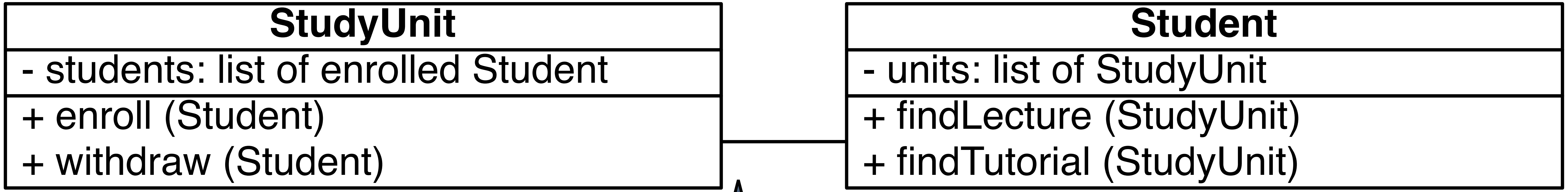
Catalog

- units : list of StudyUnit

+ relatedUnits (keyword: String) : list of StudyUnit

dashed line

Permanent relationships are modelled as association using a solid line in UML



StudyUnit

- students: list of enrolled Student

+ enroll (Student)

+ withdraw (Student)

Student

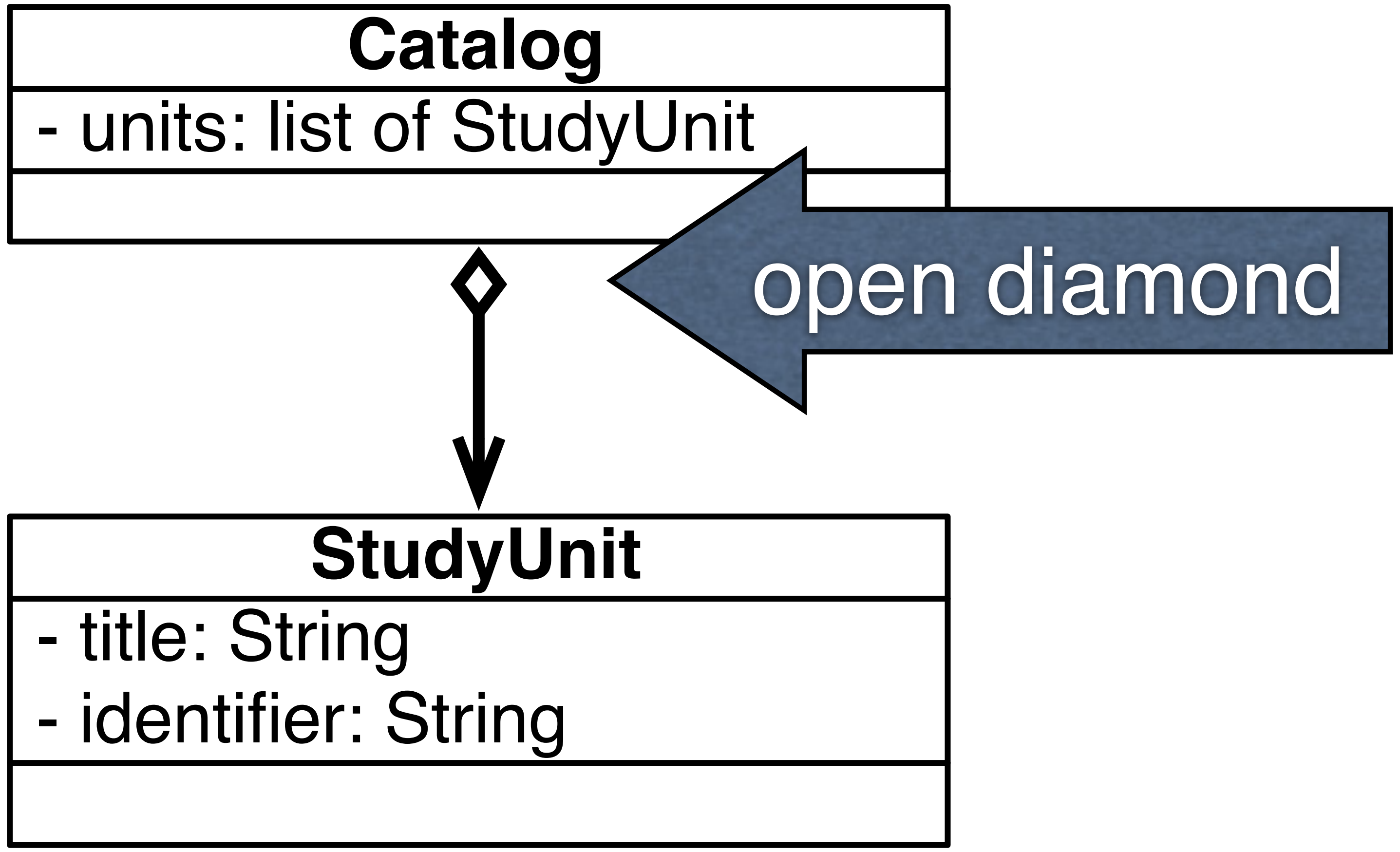
- units: list of StudyUnit

+ findLecture (StudyUnit)

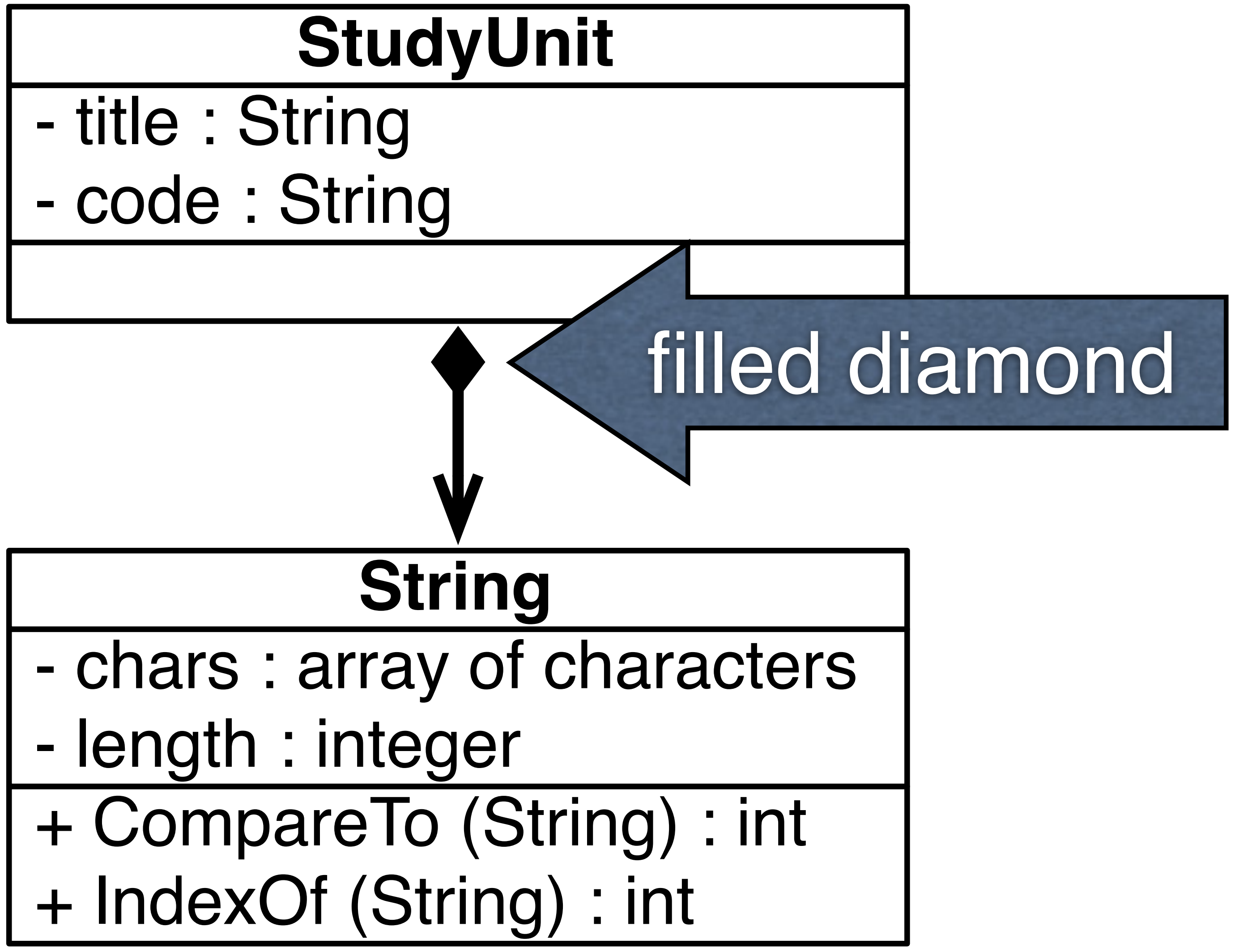
+ findTutorial (StudyUnit)

Can include arrows if relationship is in a single

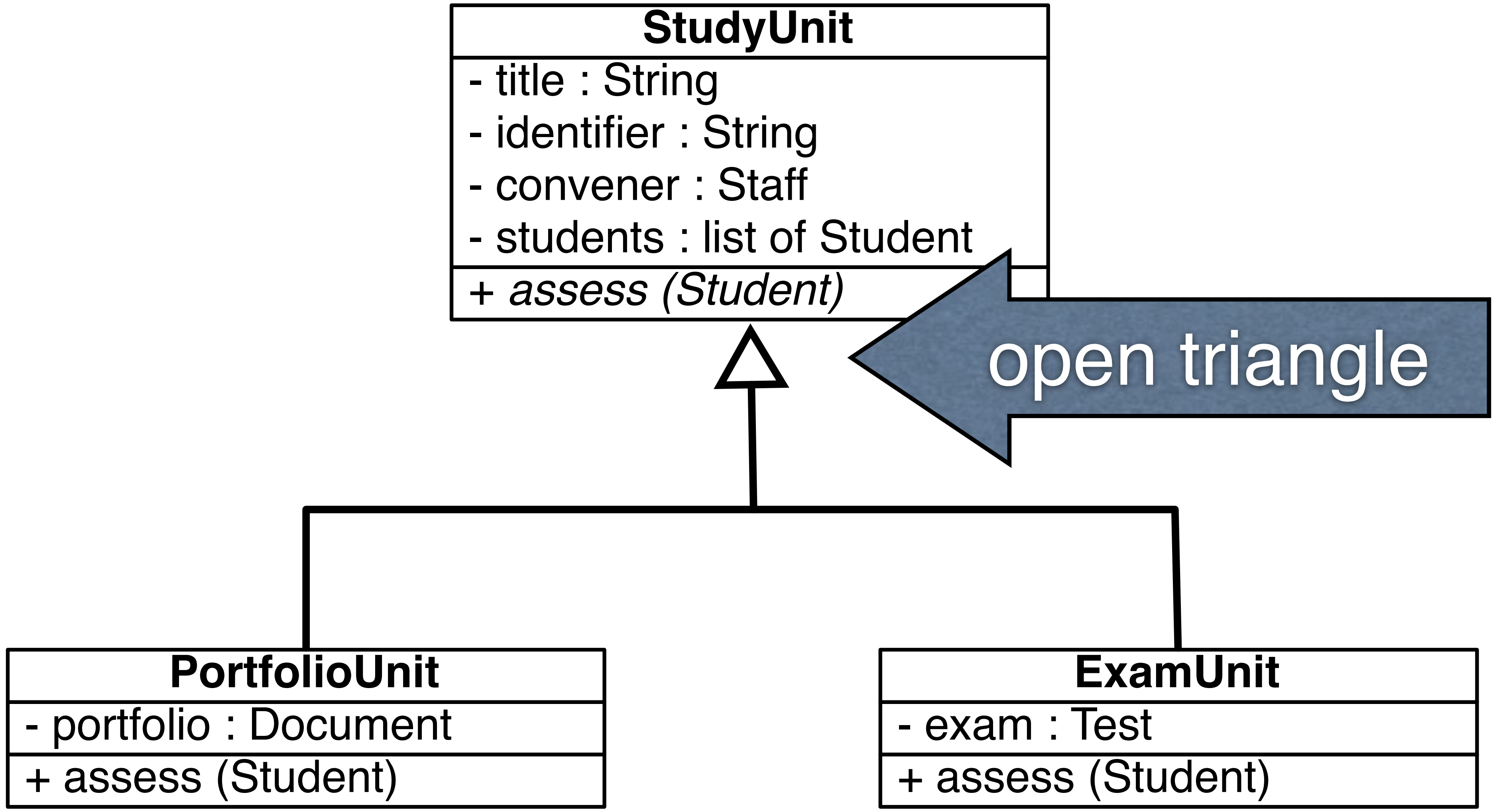
Aggregation extends association to indicate a whole-part relation



Composition is a kind of aggregation, indicating destruction of the whole involves destruction of the part



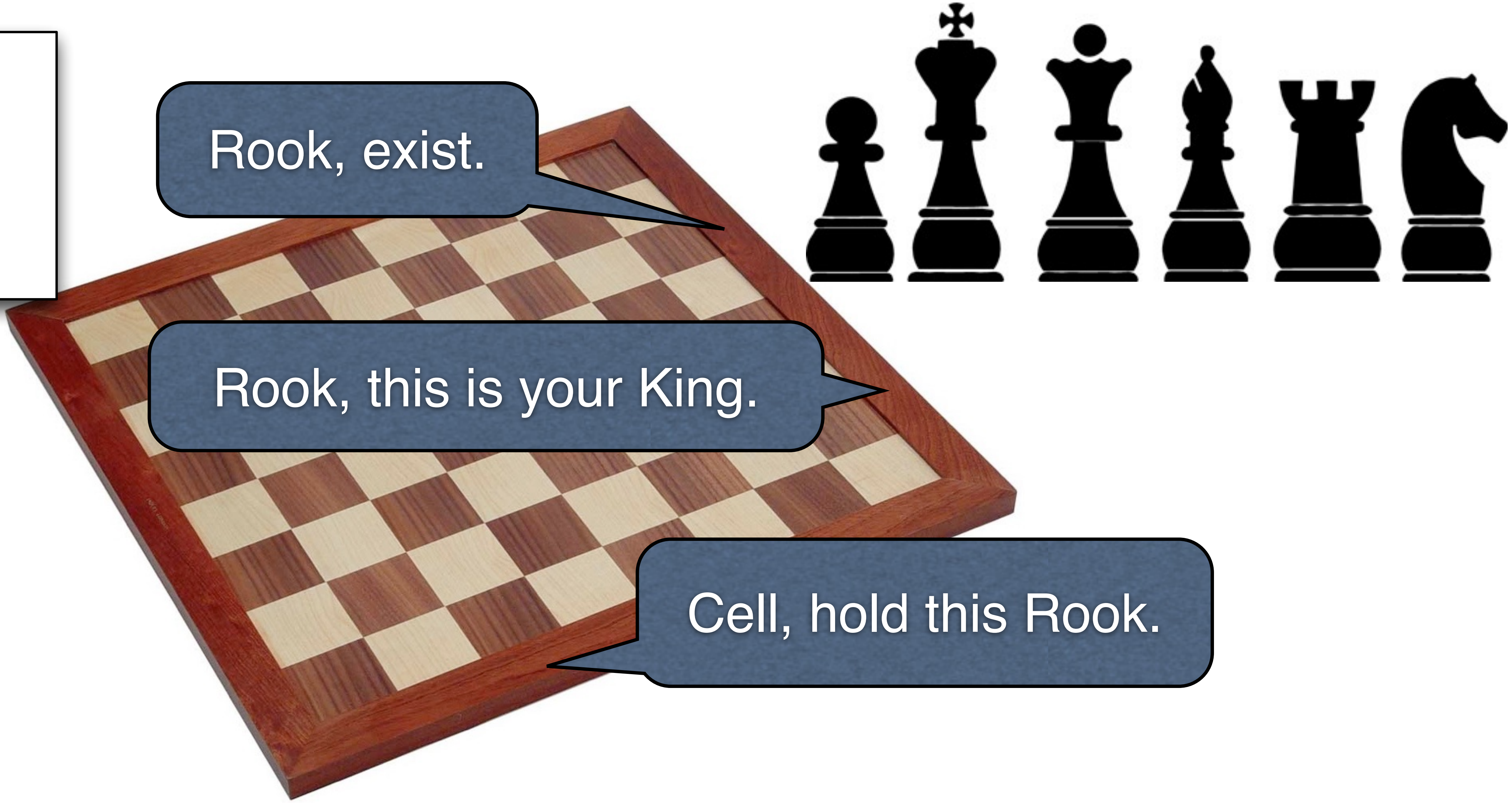
Inheritance captures class and interface inheritance for specialisation/generalisation



Use scenarios to test how your model responds to events and implements features

Chess Game

knows its Board
knows its Pieces
knows its Players
can start the game
can process a players turn



Rook, exist.

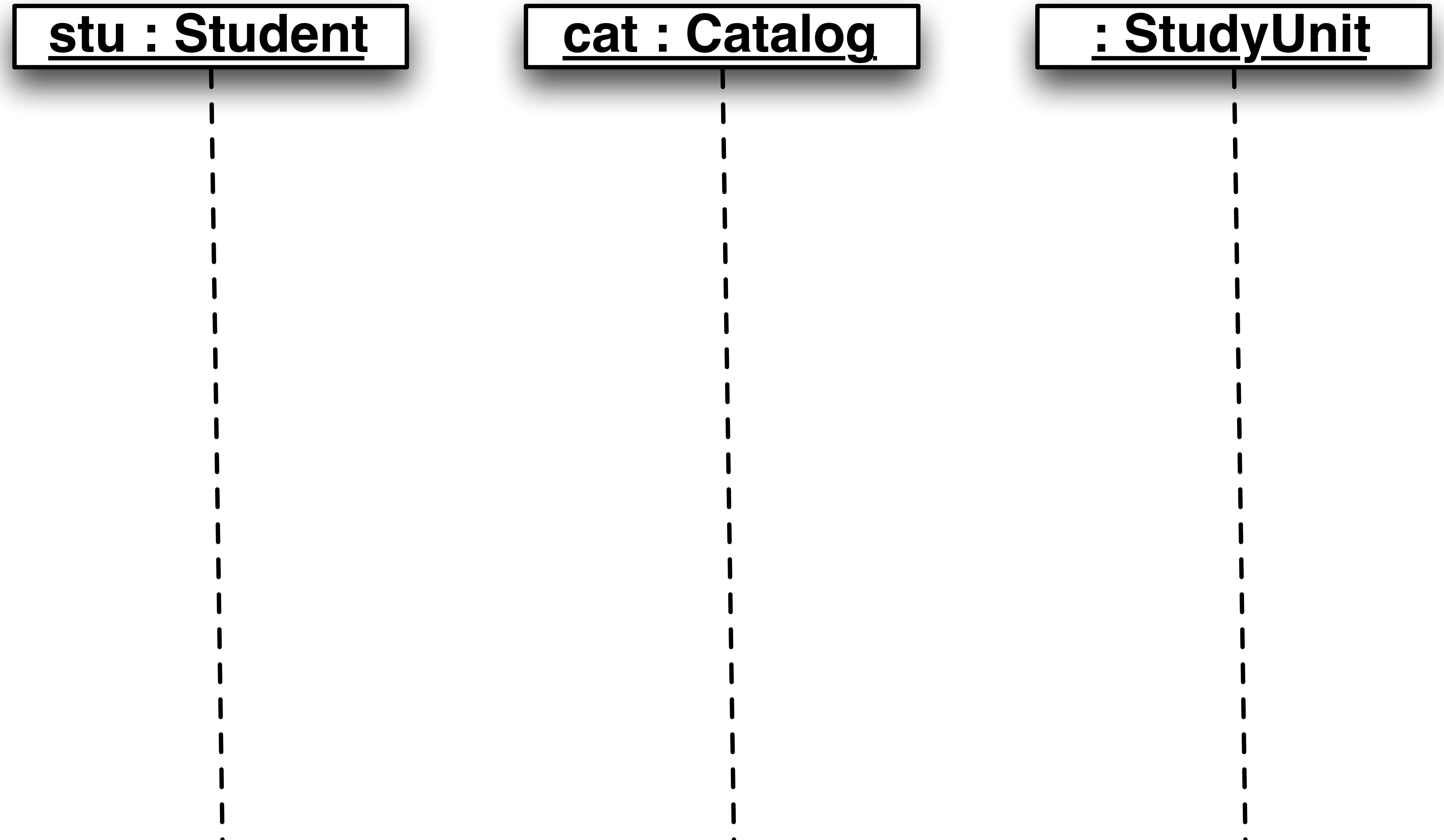
Rook, this is your King.

Board, setup.

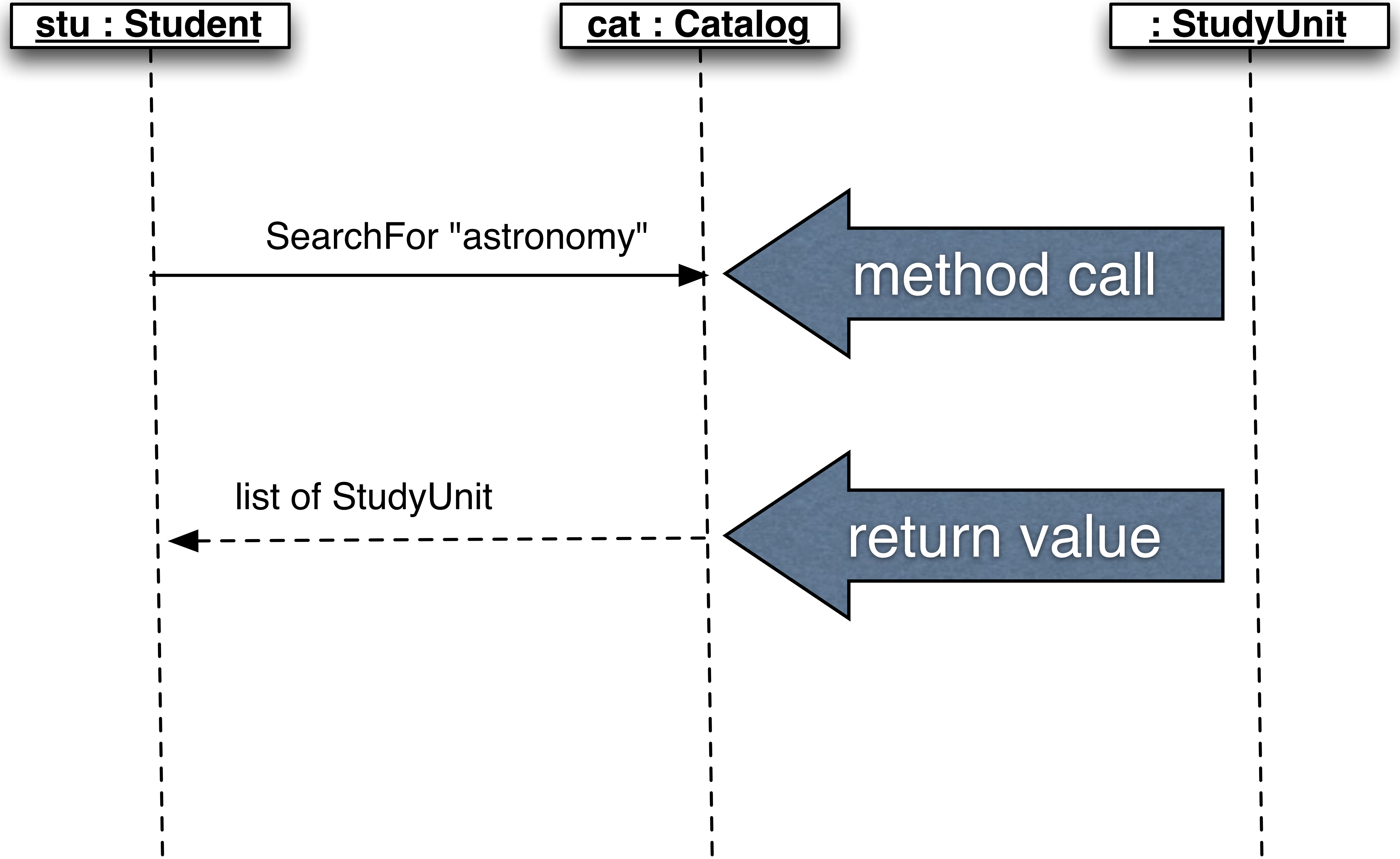
Cell, hold this Rook.

Communicate these dynamic interactions using sequence diagrams

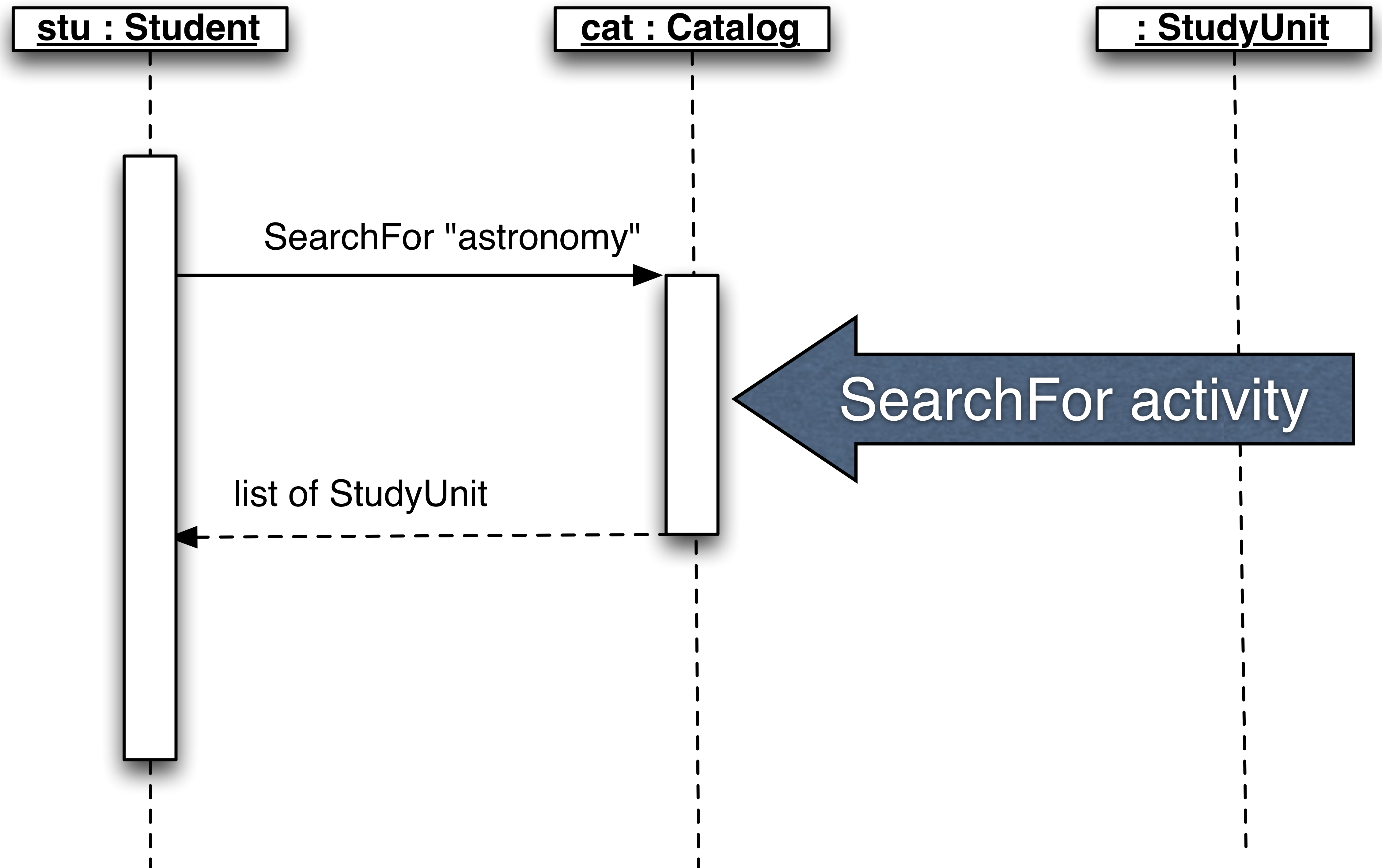
Think of sequence diagrams as scripts, with life lines defining the existence of objects



Draw arrows between lifelines to show message passing (method calls)



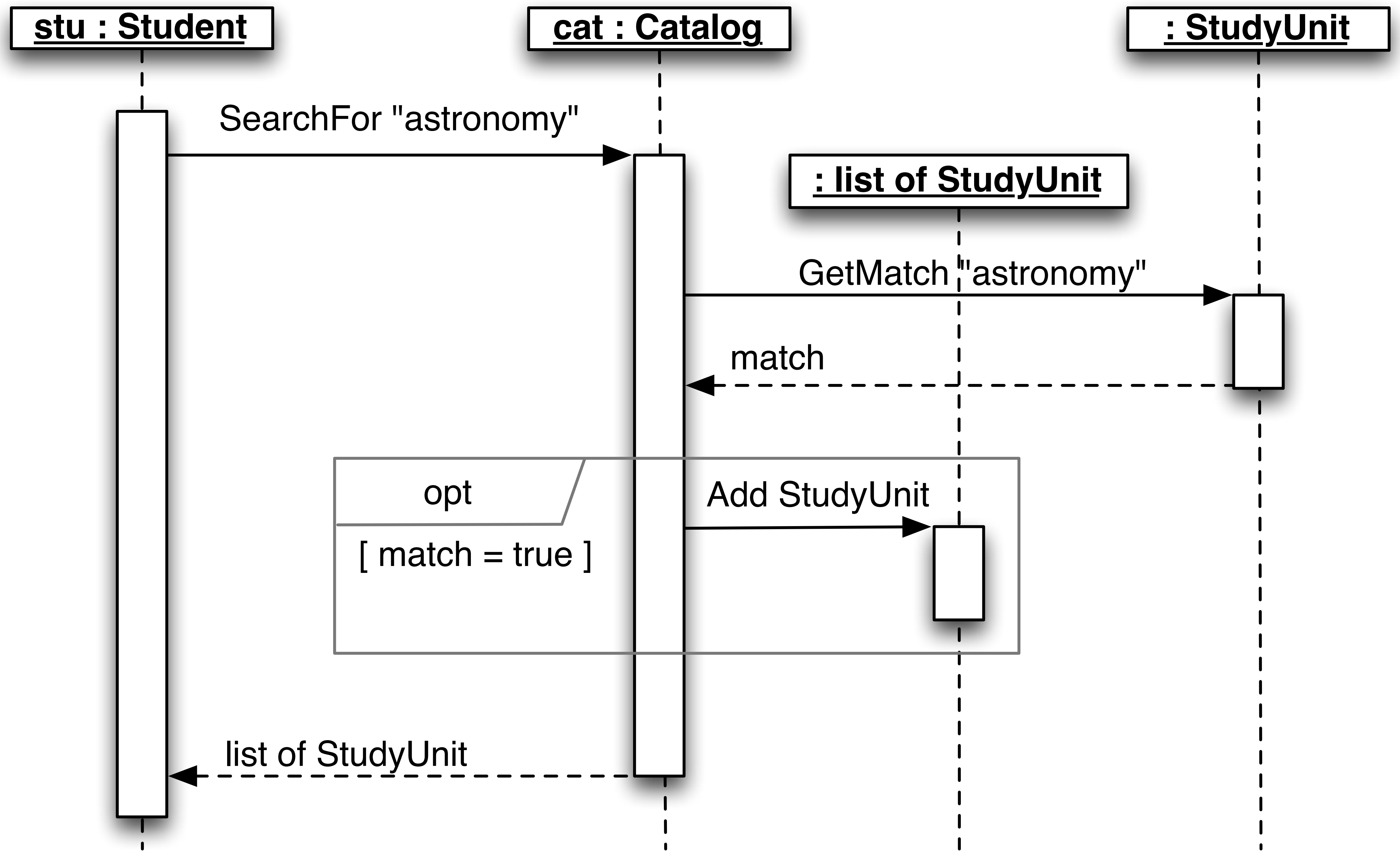
Use boxes to represent **activity**: when it is doing something or waiting for something to be done



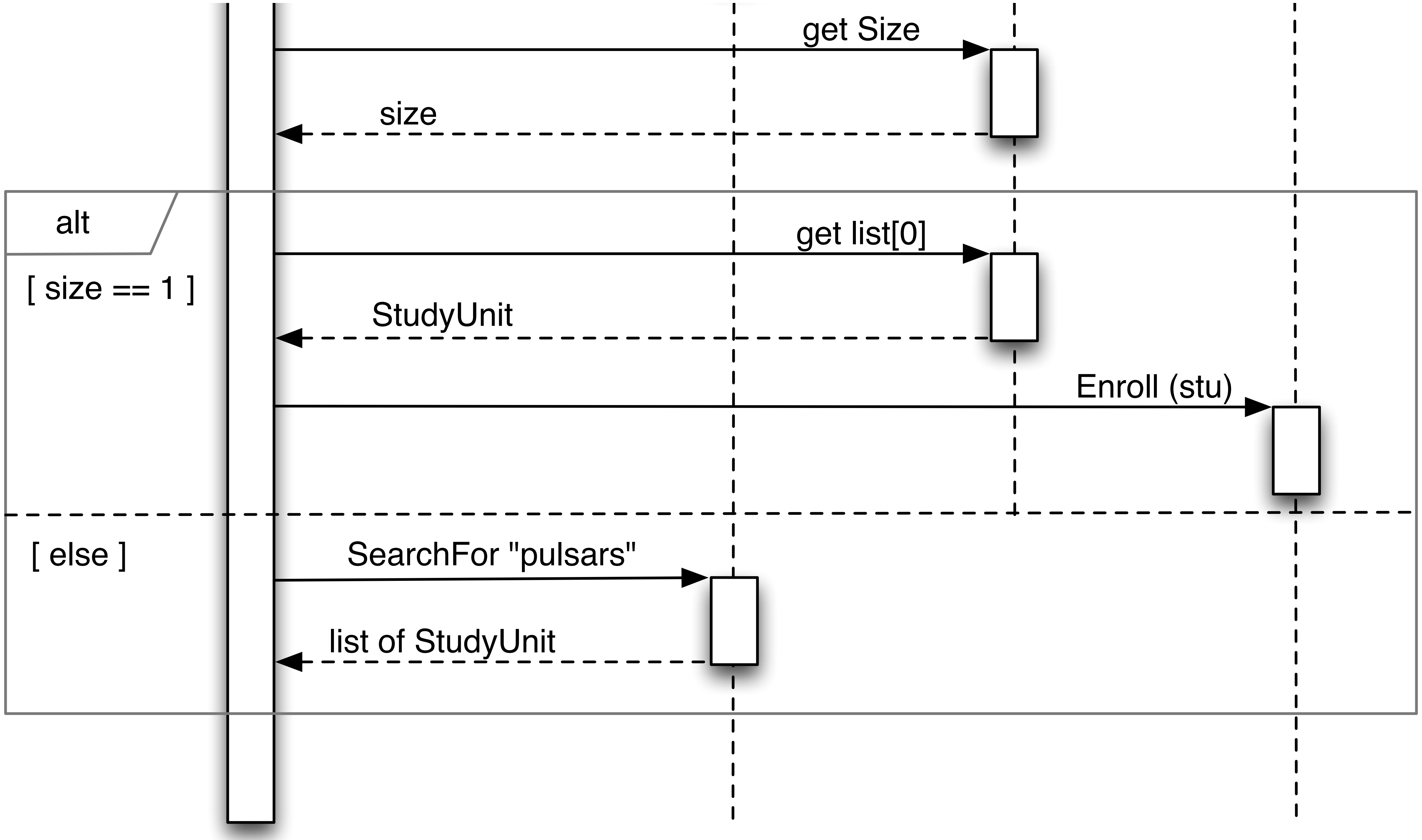
SearchFor activity

Show control flow logic using **combination fragments**

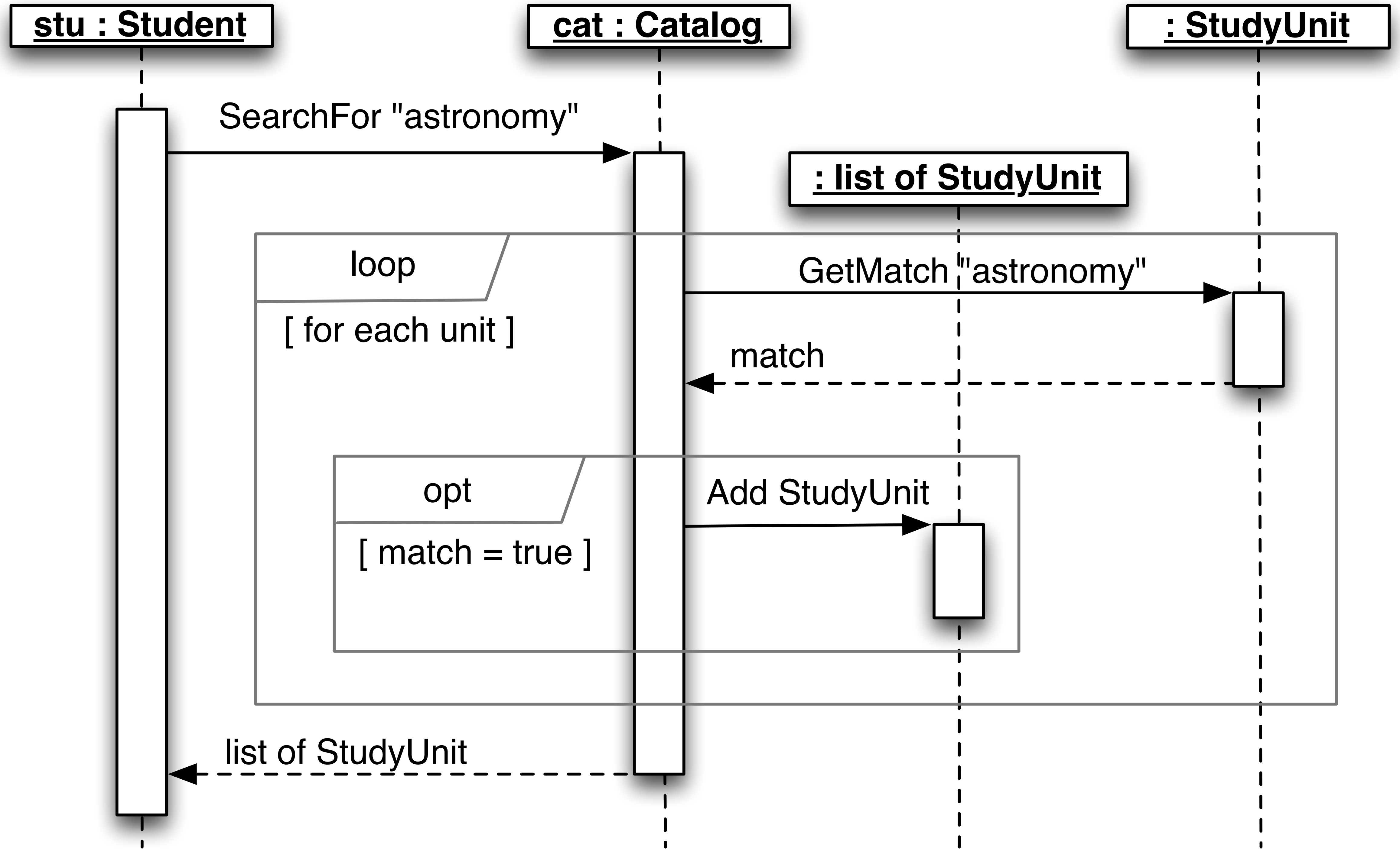
Model if using options



Show alternatives to model if with else



Use loops to model repetition



Will roles, responsibilities, and collaborations help you design object oriented programs?

Effective designs ease the process of implementation, for teams and individual

Create effective OO designs using Roles, Responsibilities, and Collaborations

Responsibility driven design focuses on object roles, responsibilities, and interactions

Roles, Responsibilities, and Collaborations