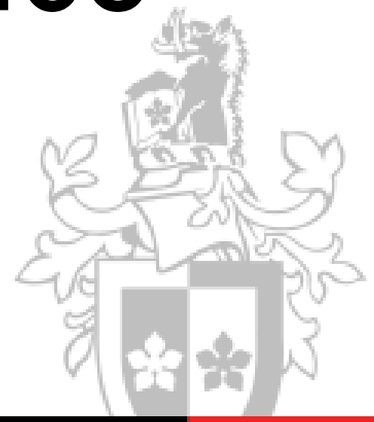
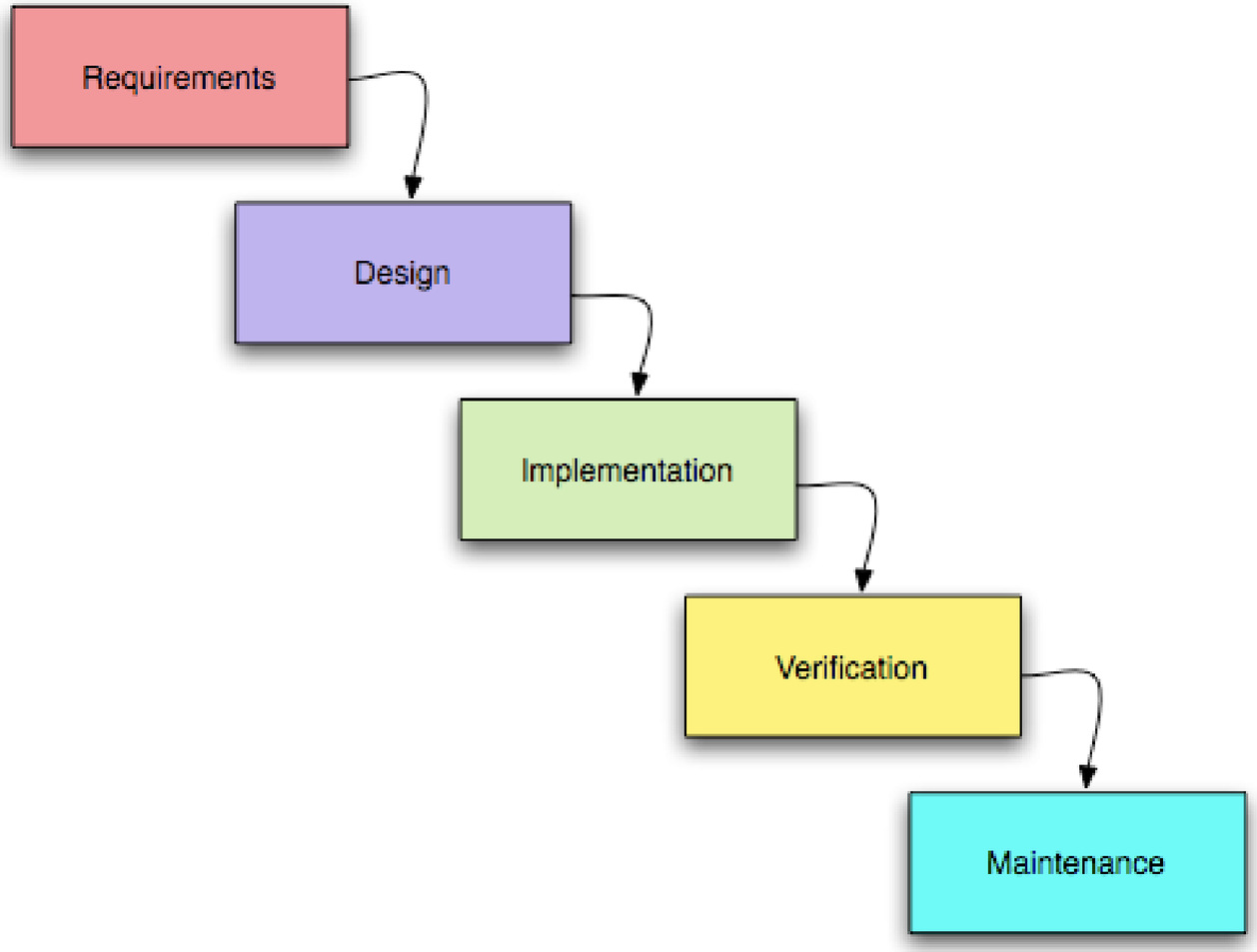
Unified Modeling Language: More on Class and Sequence Diagrams



SWINBURNE UNIVERSITY OF TECHNOLOGY

Object-oriented solutions start with analysis and design

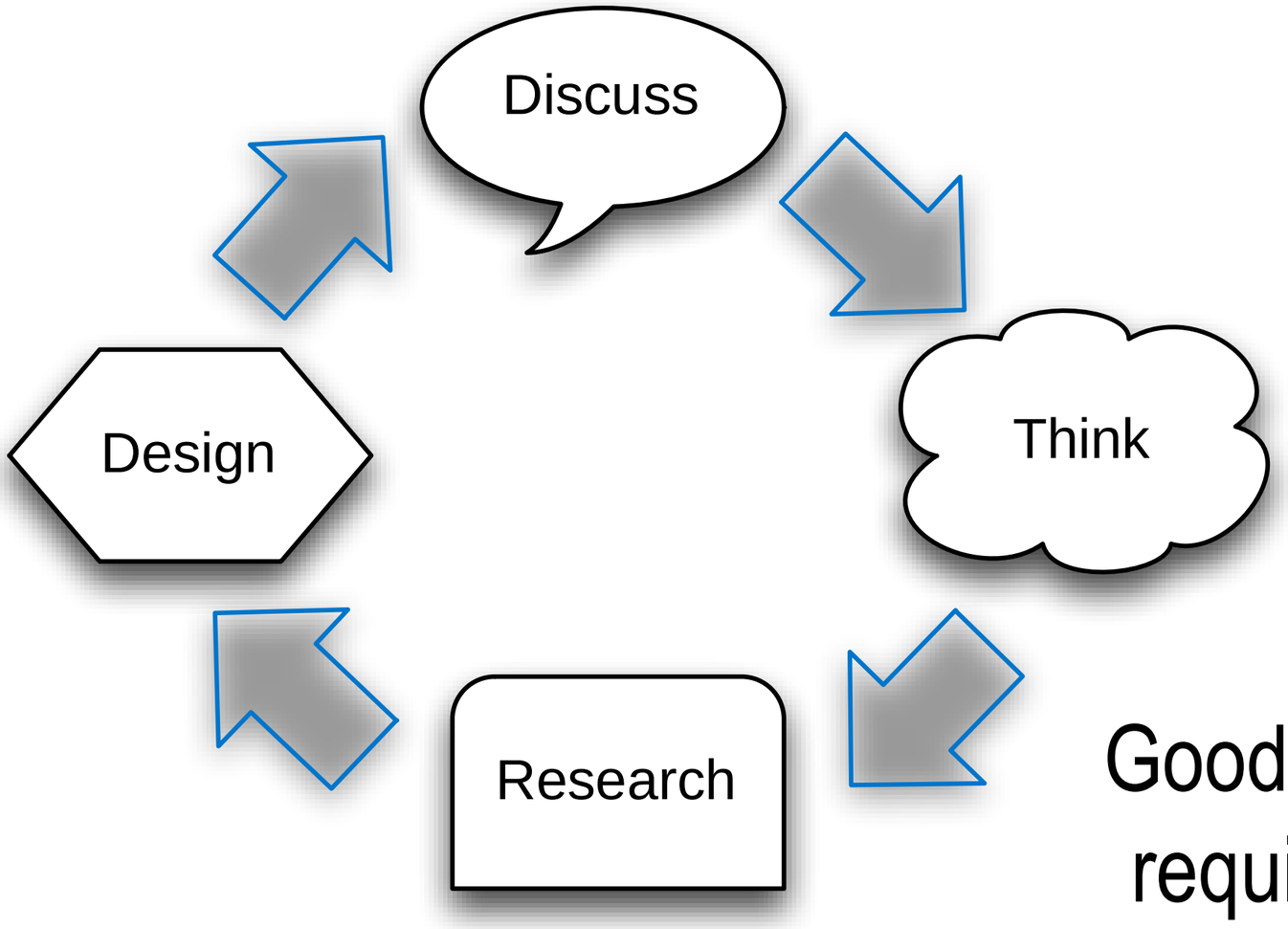


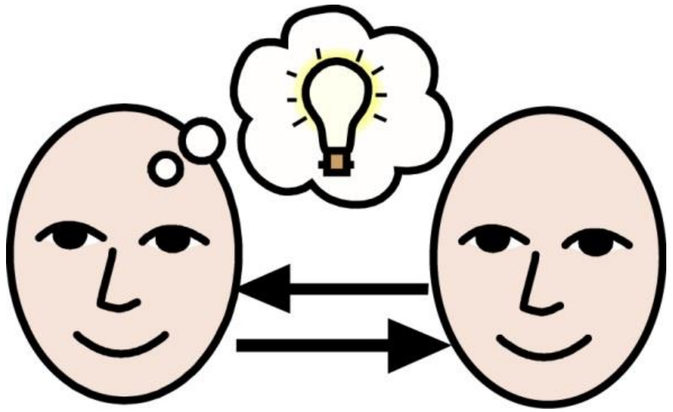
Developers must communicate with each other and other stakeholders

UML

"method for specifying, visualizing, and documenting the artifacts of an object-oriented system under development."

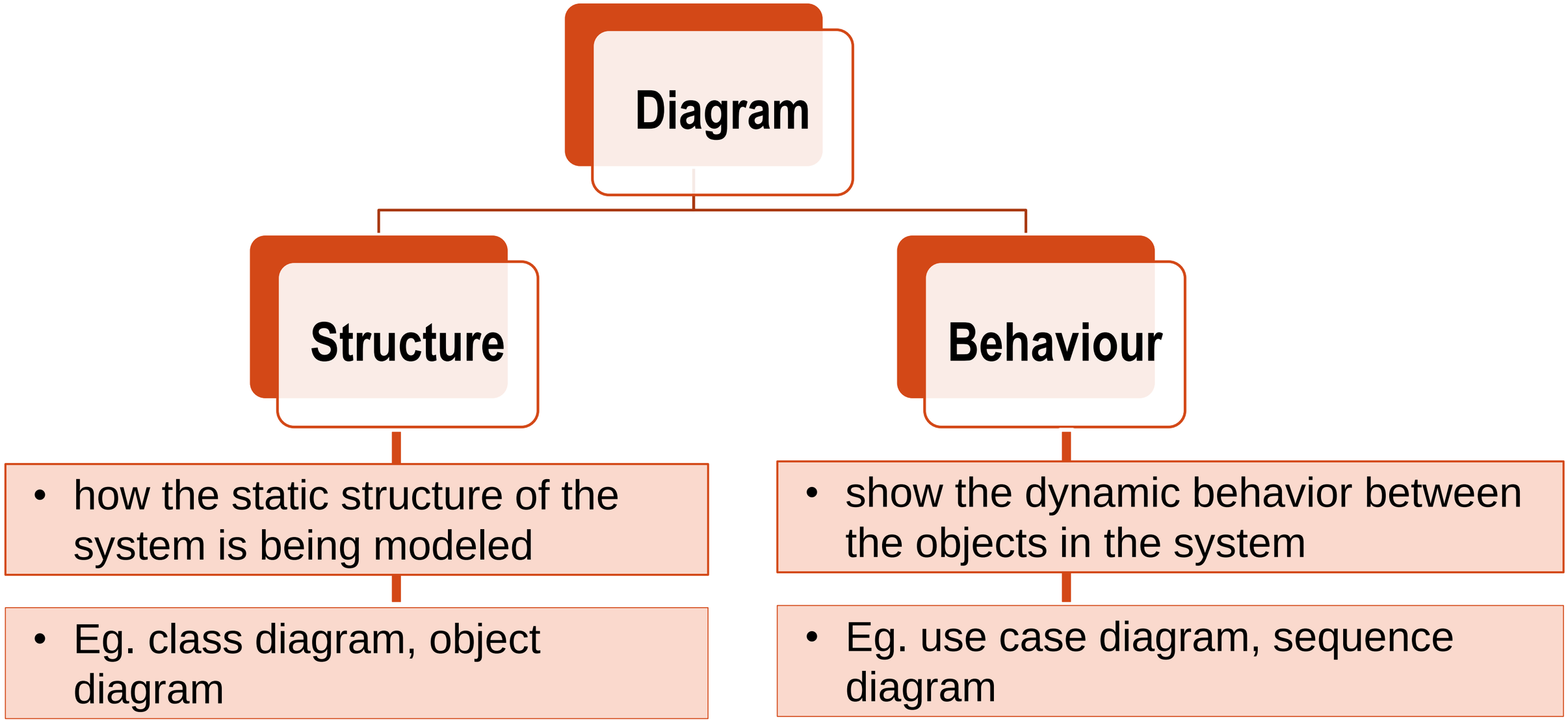
Complex solutions require multiple iterations of refinement





Good object-oriented solutions require thinking and planning

2 basic categories of diagrams



Use the Unified Modeling Language to communicate your design through diagrams

- UML diagrams increase the efficiency of design communication
- Pages of text are inefficient design tools
- Communicate patterns and ideas through meaningful symbols
- Accessible to all stakeholders (domain experts, developers & end users)

Modelling guidelines

- Appropriate abstractions must be defined
- Roles and responsibilities must be factored into classes with appropriate granularity

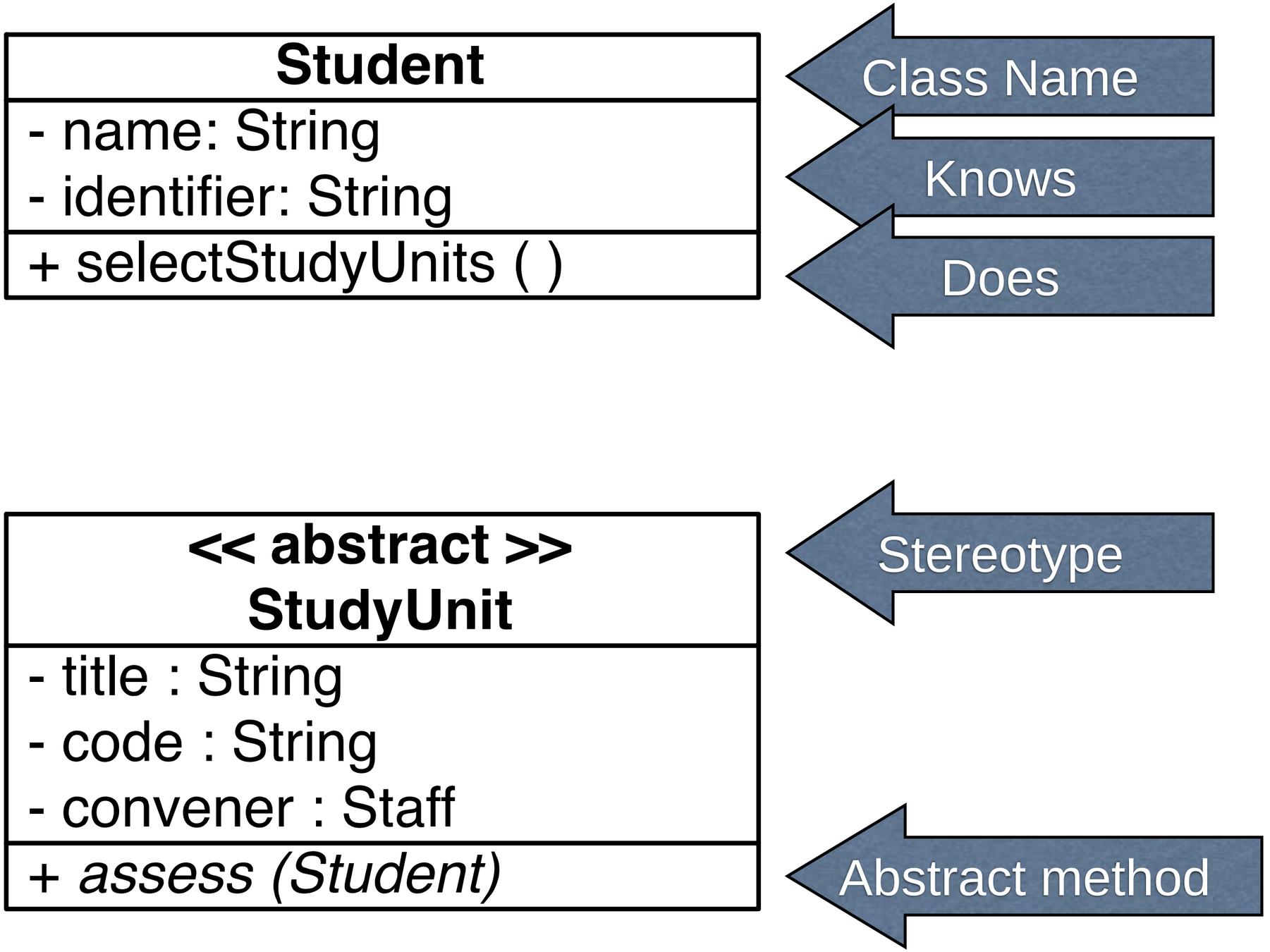
  too small = too generic - does not reduce complexity

  too large = too problem-specific - cannot be reused

Types being modeled

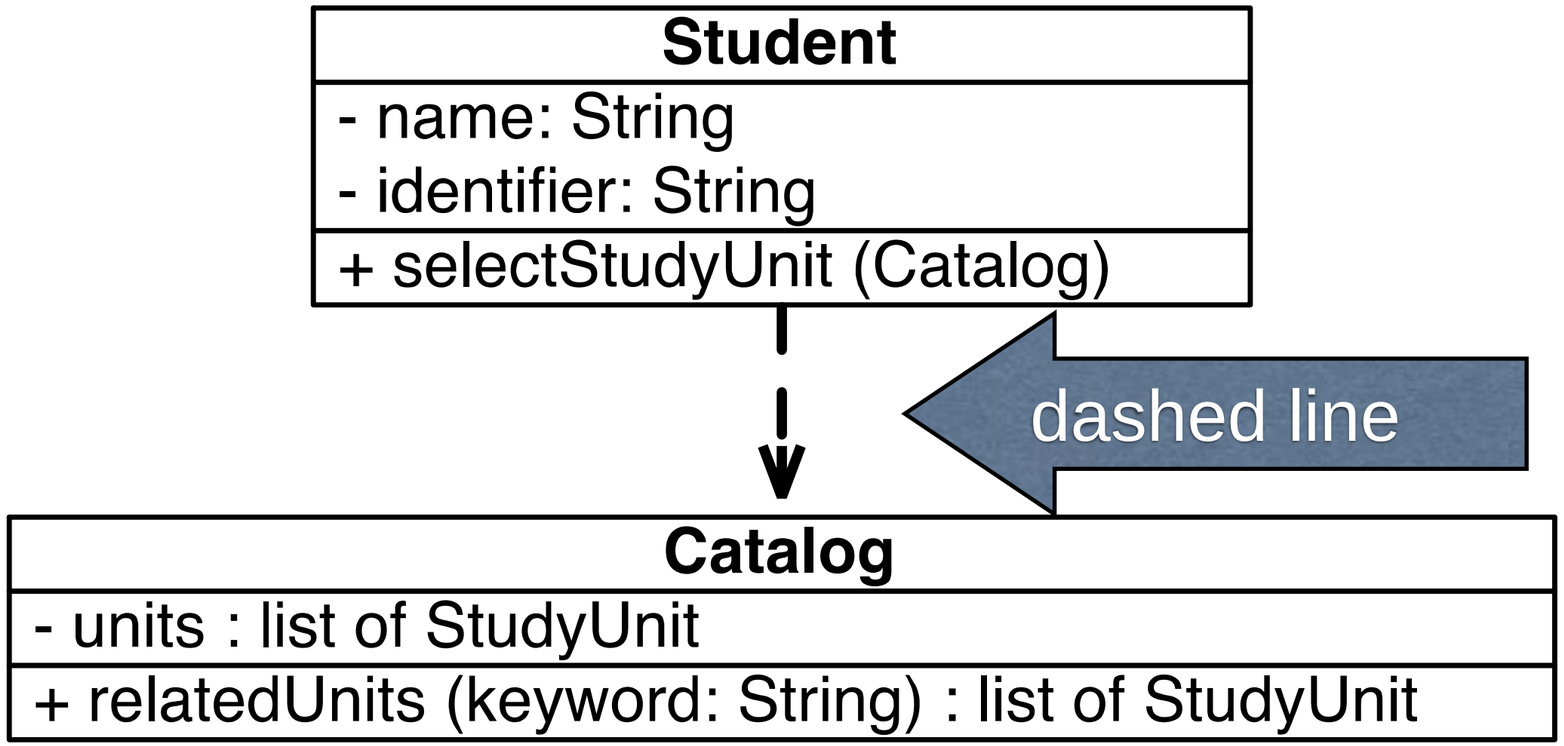
- Class
- Object
- Abstract class
- Data type

Responsibilities identify
what an object knows and does

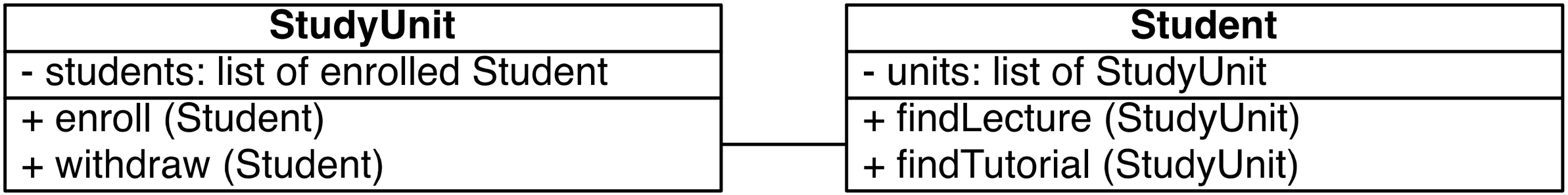

Student
- name: String
- identifier: String
+ selectStudyUnits ( )
Class Name
Knows
Does
<< abstract >>
StudyUnit
- title : String
- code : String
- convener : Staff
+ assess (Student)
Stereotype
Abstract method

Relationships identify dependencies between objects

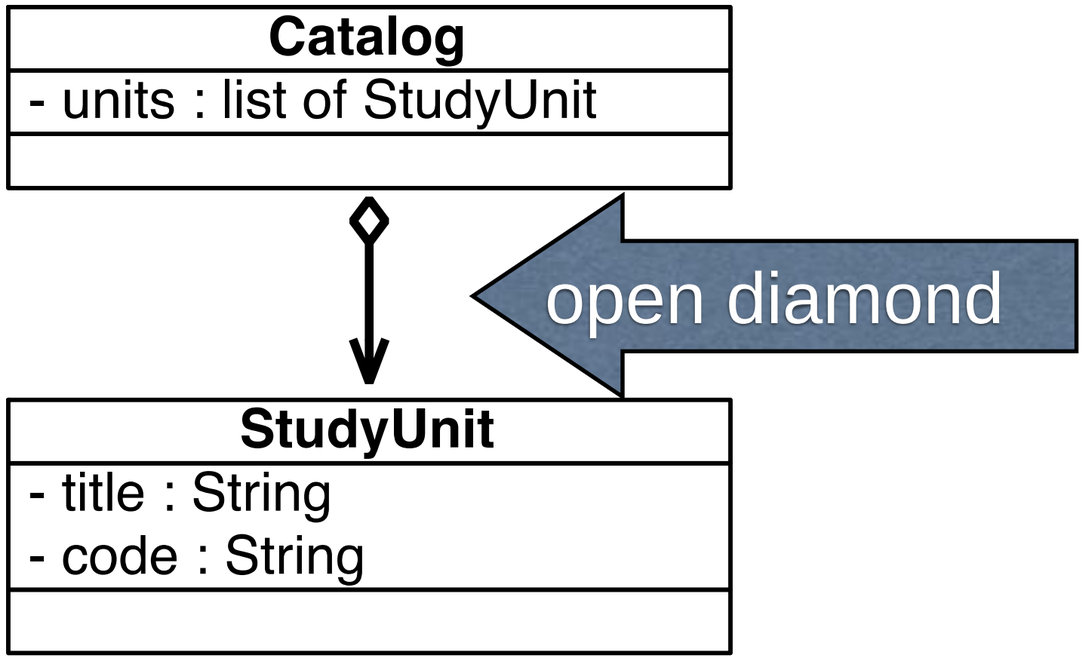
Dependence


Student
- name: String
- identifier: String
+ selectStudyUnit (Catalog)
dashed line
Catalog
- units : list of StudyUnit
+ relatedUnits (keyword: String) : list of StudyUnit

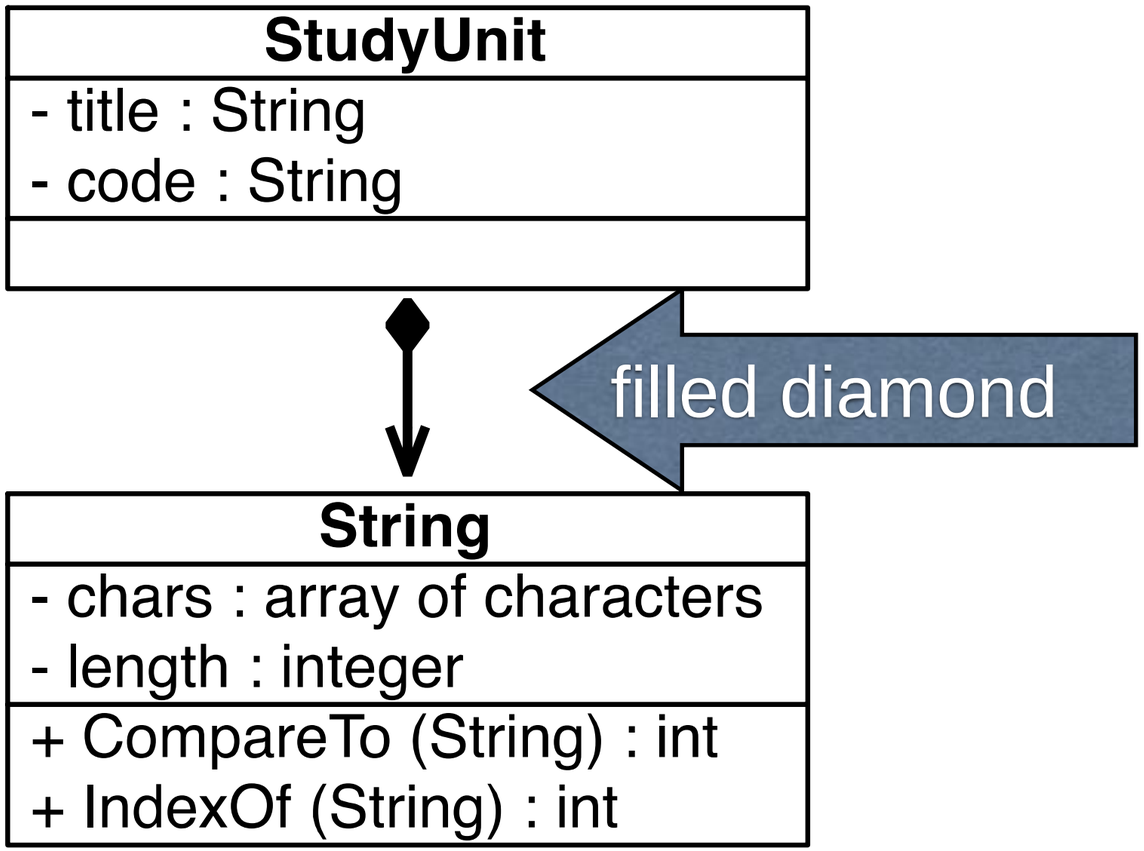
Association



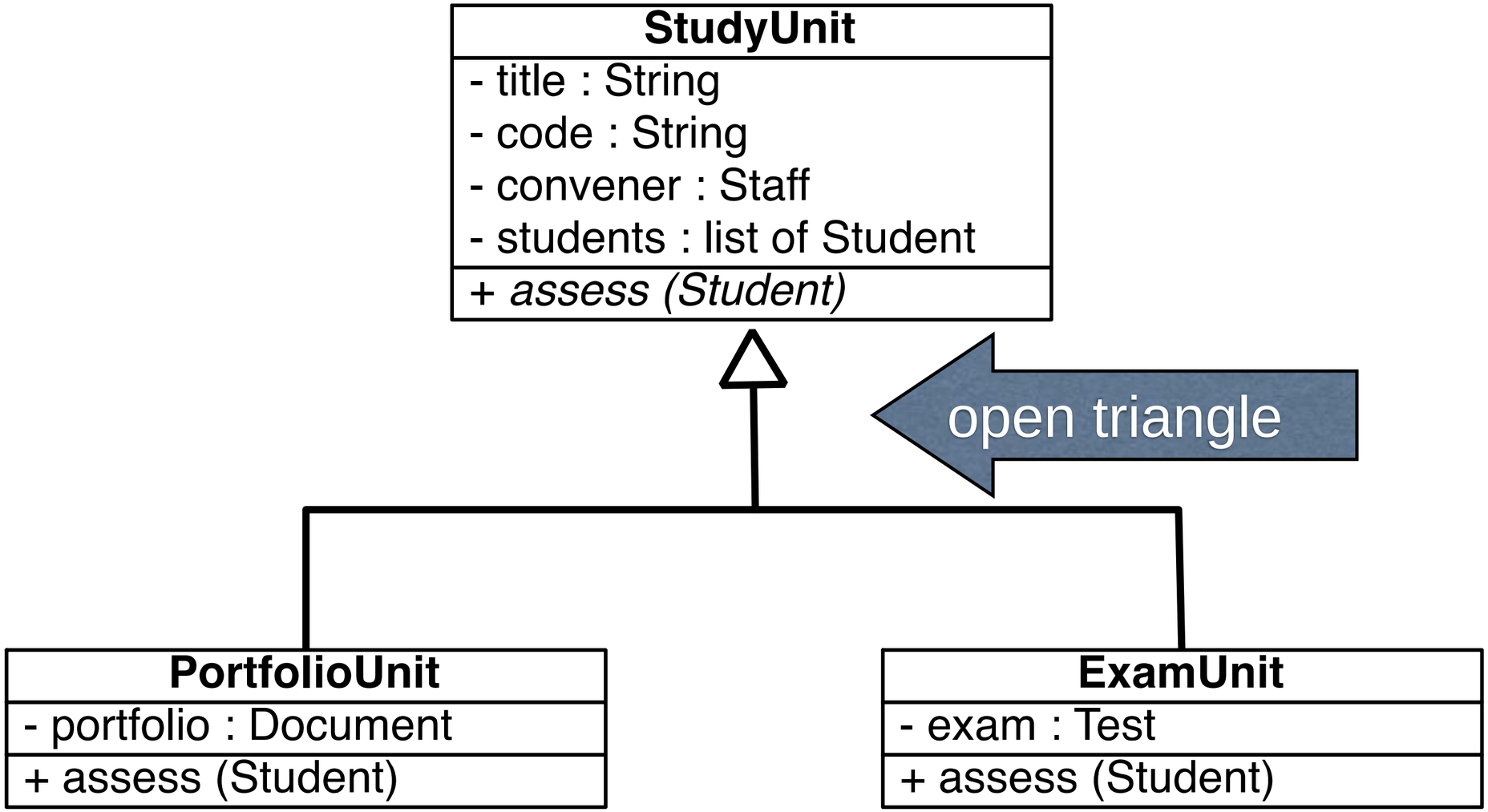
Aggregation



Composition



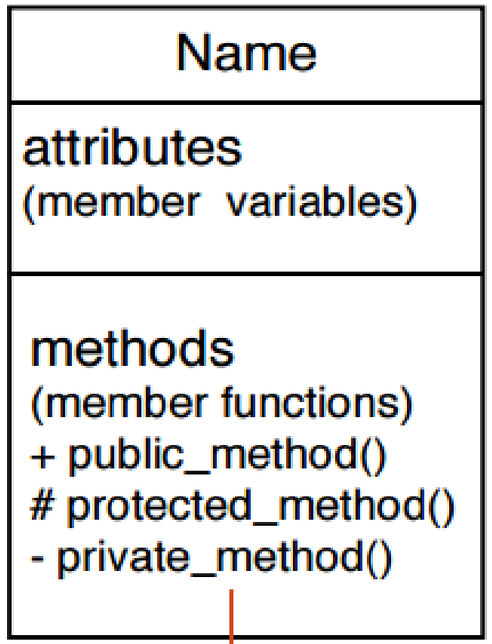
Inheritance



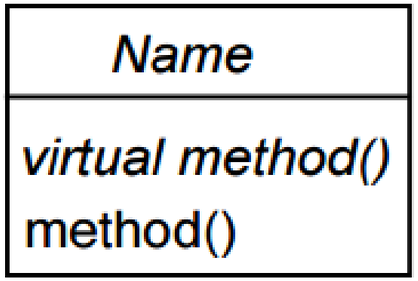
Let’s have a look on some examples!

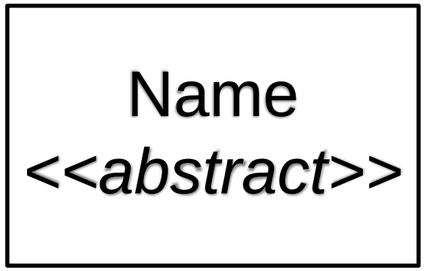
Types

Class

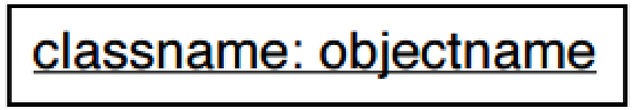


Abstract class

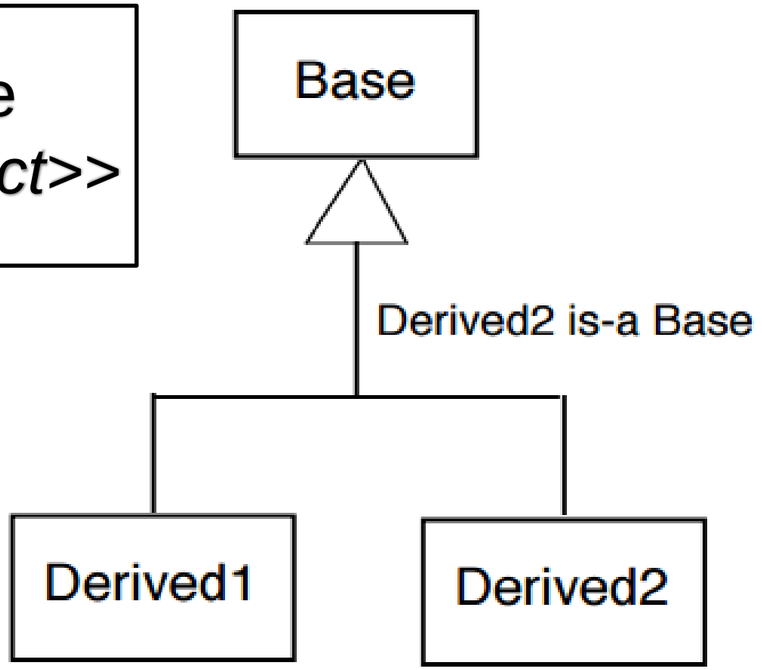




Object



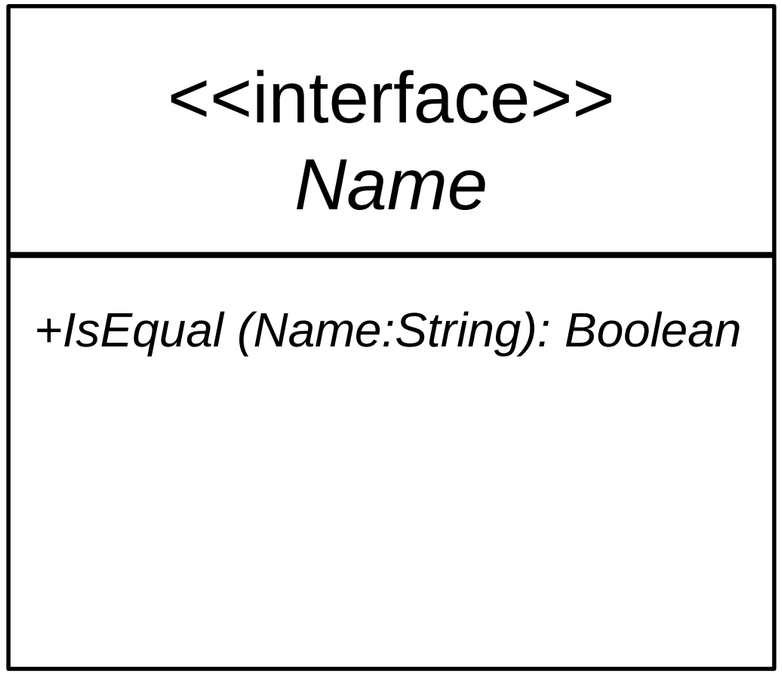
Inheritance (is-a) relationship



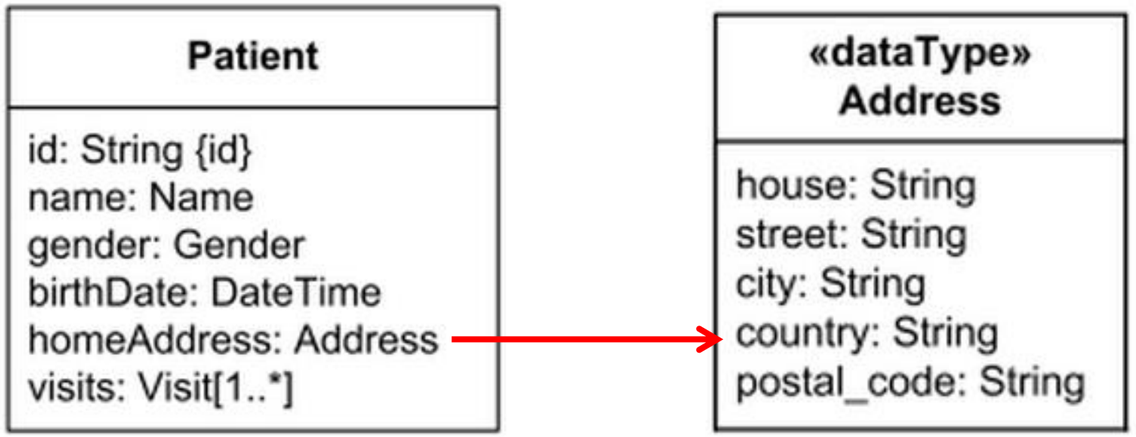
- name(parameter list) : type of value returned
- getFlightDuration(flightNo: String) : int

Types

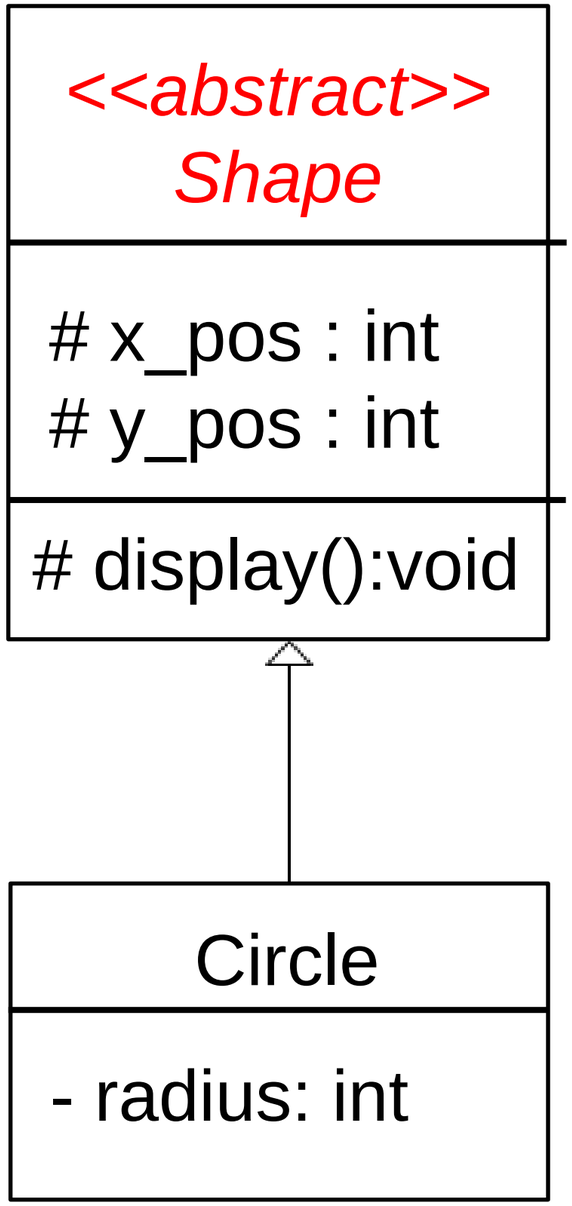
Interface



Data type



Example 1



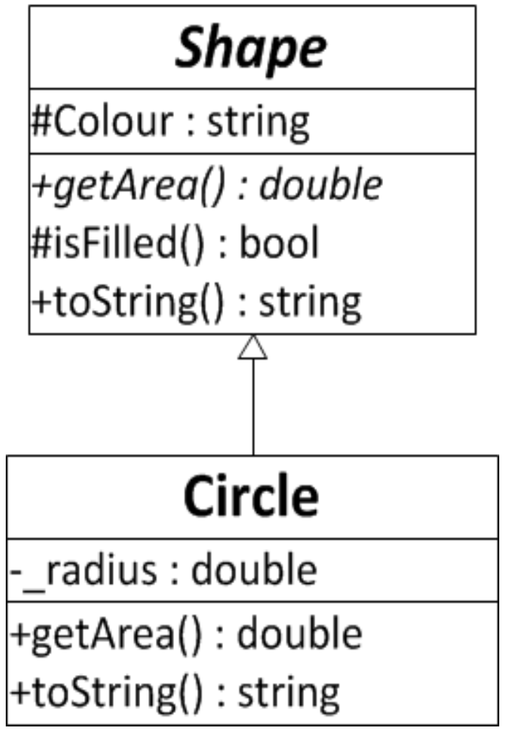
Base class

```
public abstract class Shape
{
    protected int x_pos;
    protected int y_pos;

    protected void display()
    {
        Console.WriteLine("Shape: ");
    }
}
```

Derived class

```
class Circle : Shape {
    private int radius;
}
```

Example 2



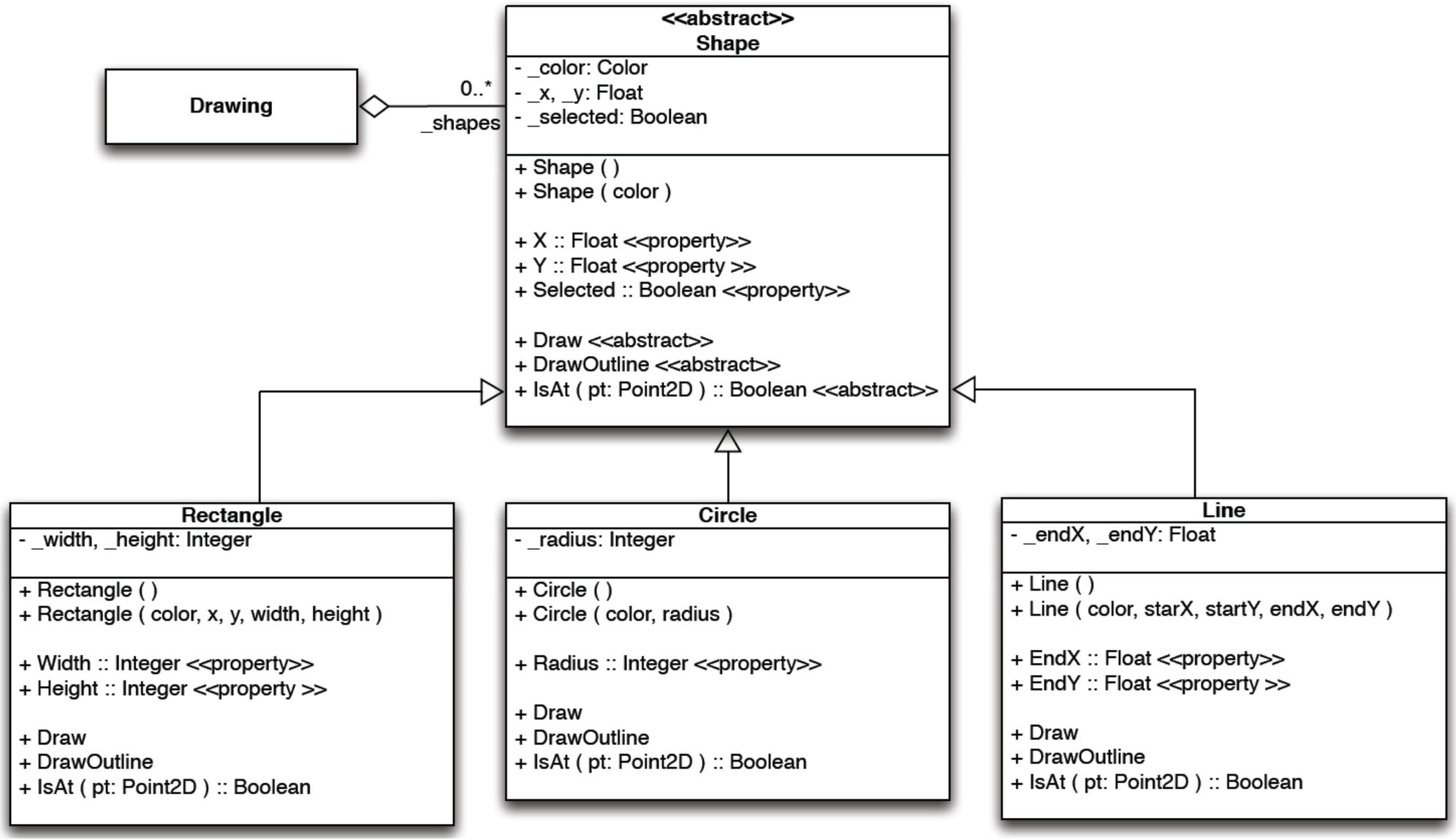
Base class

```
public abstract class Shape
{
    protected string Colour;
    public abstract double getArea();
    protected bool isFilled()
    {
        ....
    }
    public virtual string toString()    {
        Console.WriteLine("Colour: {0} ", Colour);

    }
}
```

Derived class

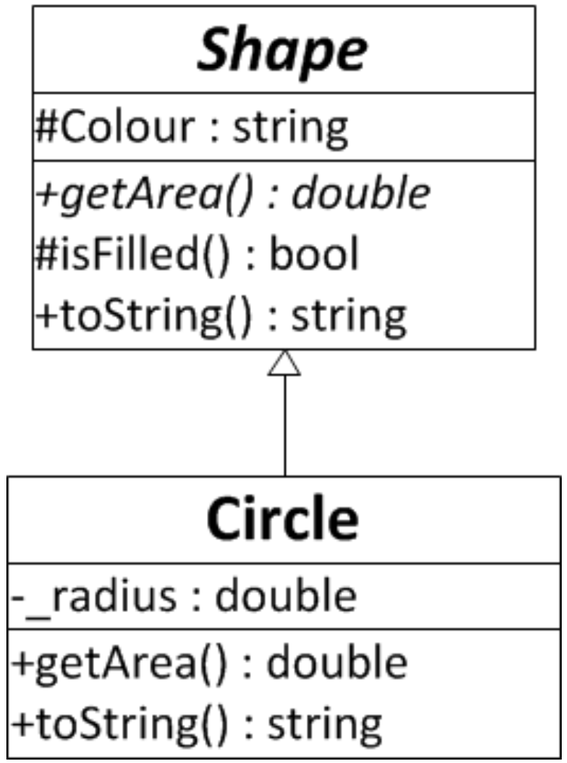
```
class Circle: Shape{

        private double _radius;

        public override double getArea(){

            return (3.14 * Math.Pow(_radius,
2);     }

    protected override string toString(){
        Console.WriteLine("Area: {0} ",
        getArea());

  }

}
```

Example 3


Drawing
0..*
_shapes
<<abstract>>
Shape
- _color: Color
- _x, _y: Float
- _selected: Boolean
+ Shape ( )
+ Shape (color)
+ X :: Float <<property>>
+ Y :: Float <<property >>
+ Selected :: Boolean <<property>>
+ Draw <<abstract>>
+ DrawOutline <<abstract>>
+ IsAt (pt: Point2D) :: Boolean <<abstract>>
Rectangle
- _width, _height: Integer
+ Rectangle ( )
+ Rectangle (color, x, y, width, height)
+ Width :: Integer <<property>>
+ Height :: Integer <<property >>
+ Draw
+ DrawOutline
+ IsAt (pt: Point2D) :: Boolean
Circle
- _radius: Integer
+ Circle ( )
+ Circle (color, radius)
+ Radius :: Integer <<property>>
+ Draw
+ DrawOutline
+ IsAt (pt: Point2D) :: Boolean
Line
- _endX, _endY: Float
+ Line ( )
+ Line (color, starX, startY, endX, endY)
+ EndX :: Float <<property>>
+ EndY :: Float <<property >>
+ Draw
+ DrawOutline
+ IsAt (pt: Point2D) :: Boolean

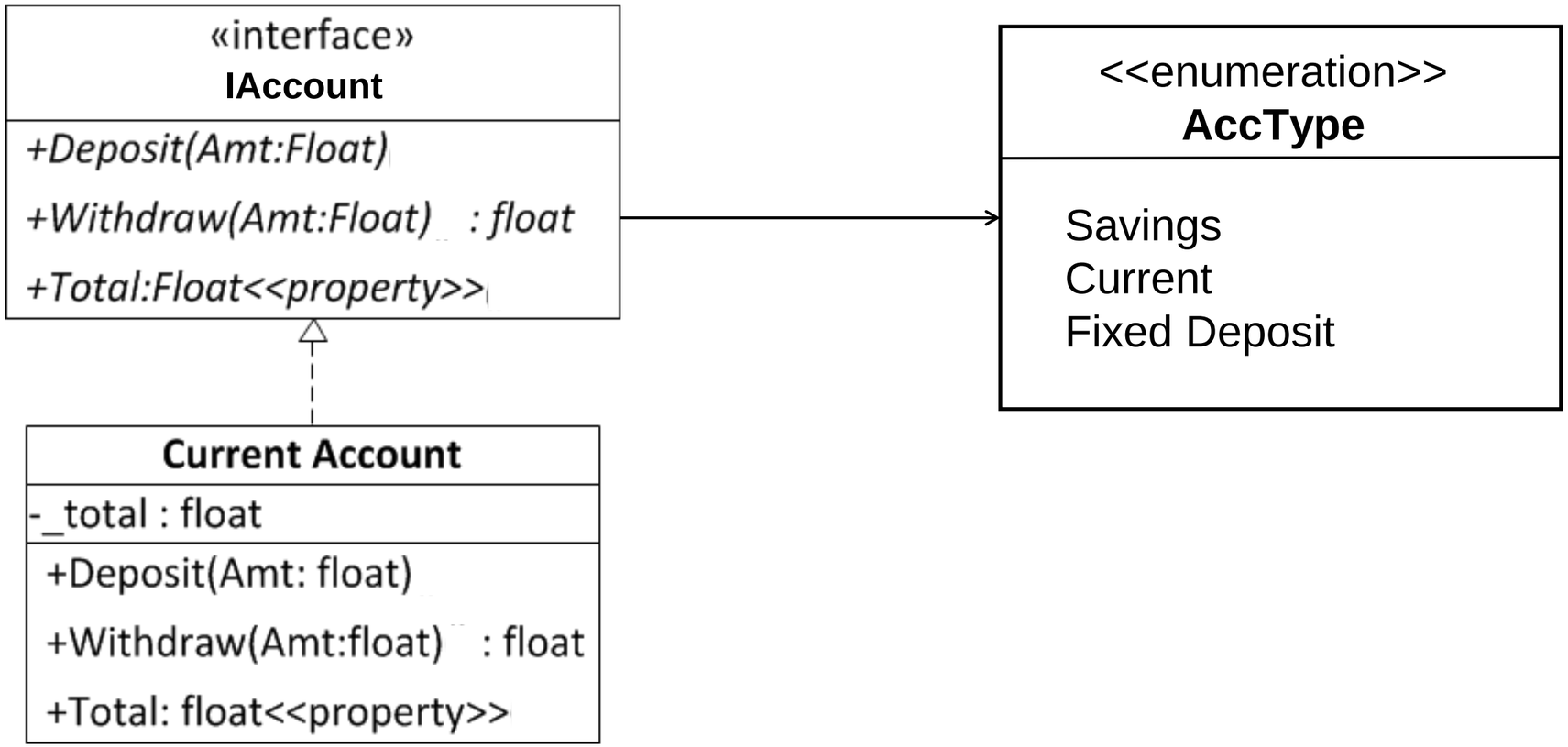
Activity 1

- Identify the object-oriented concepts applied in example 2
- Indicate where they are being implemented



Activity 2

- Write the skeleton codes for the following diagram

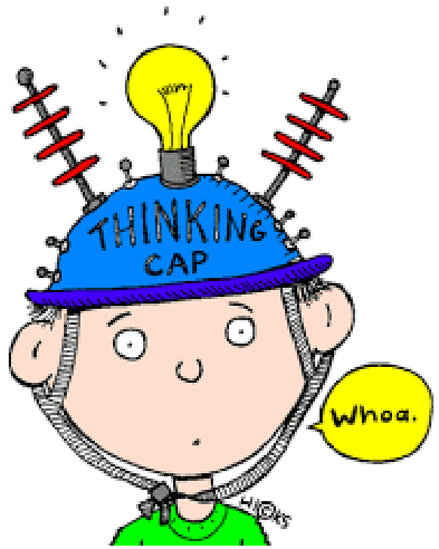


Activity 3

Time to put on your thinking cap

Refine the class diagram in Activity 2 to include

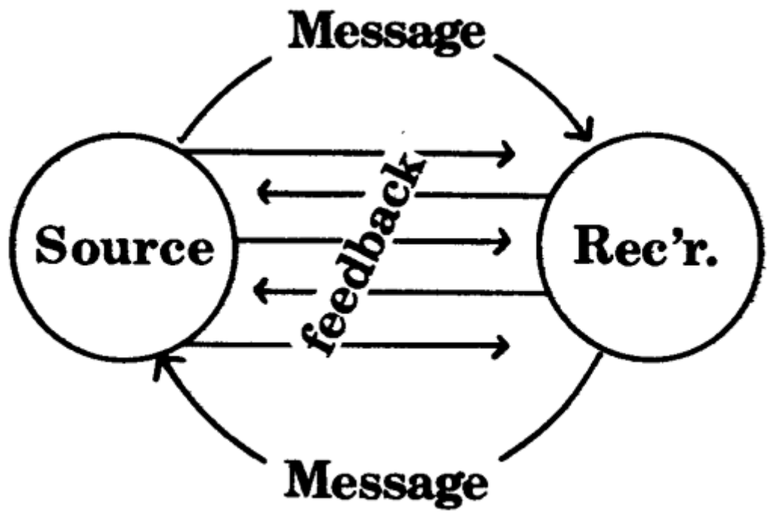
- Savings Account class
- CRUD operations



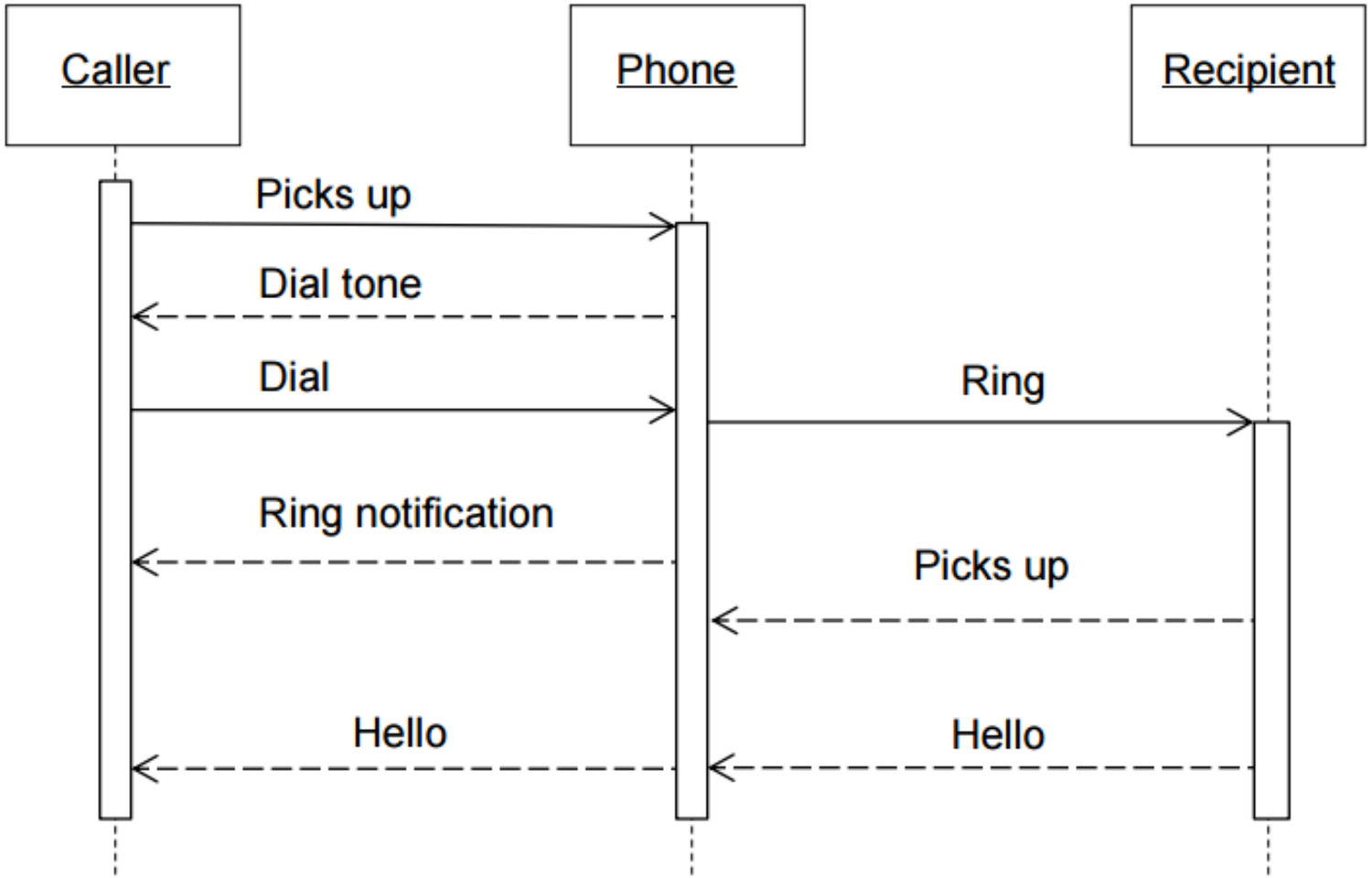
UML Sequence Diagrams

- UML Sequence Diagrams show interactions between objects in the sequential order.
- Dynamic modelling

It is analogous to a script for a play.



Let's have some basic idea on Sequence Diagram (make a phone call)

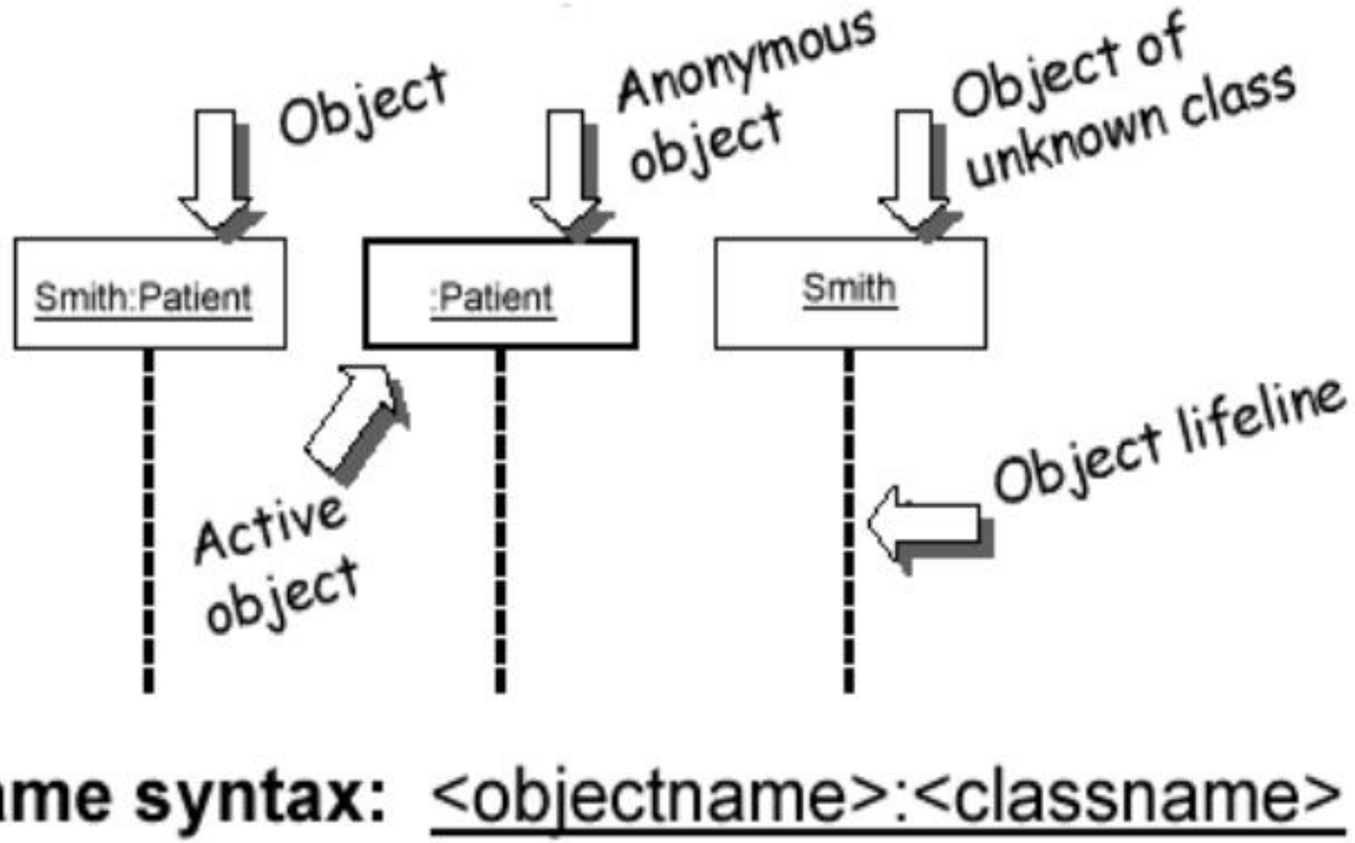


Resource: http://csis.pace.edu/~marchese/CS389/L9/Sequence%20Diagram%20Tutorial.pdf

Representing Objects

Squares with object type, optionally preceded by "name :"

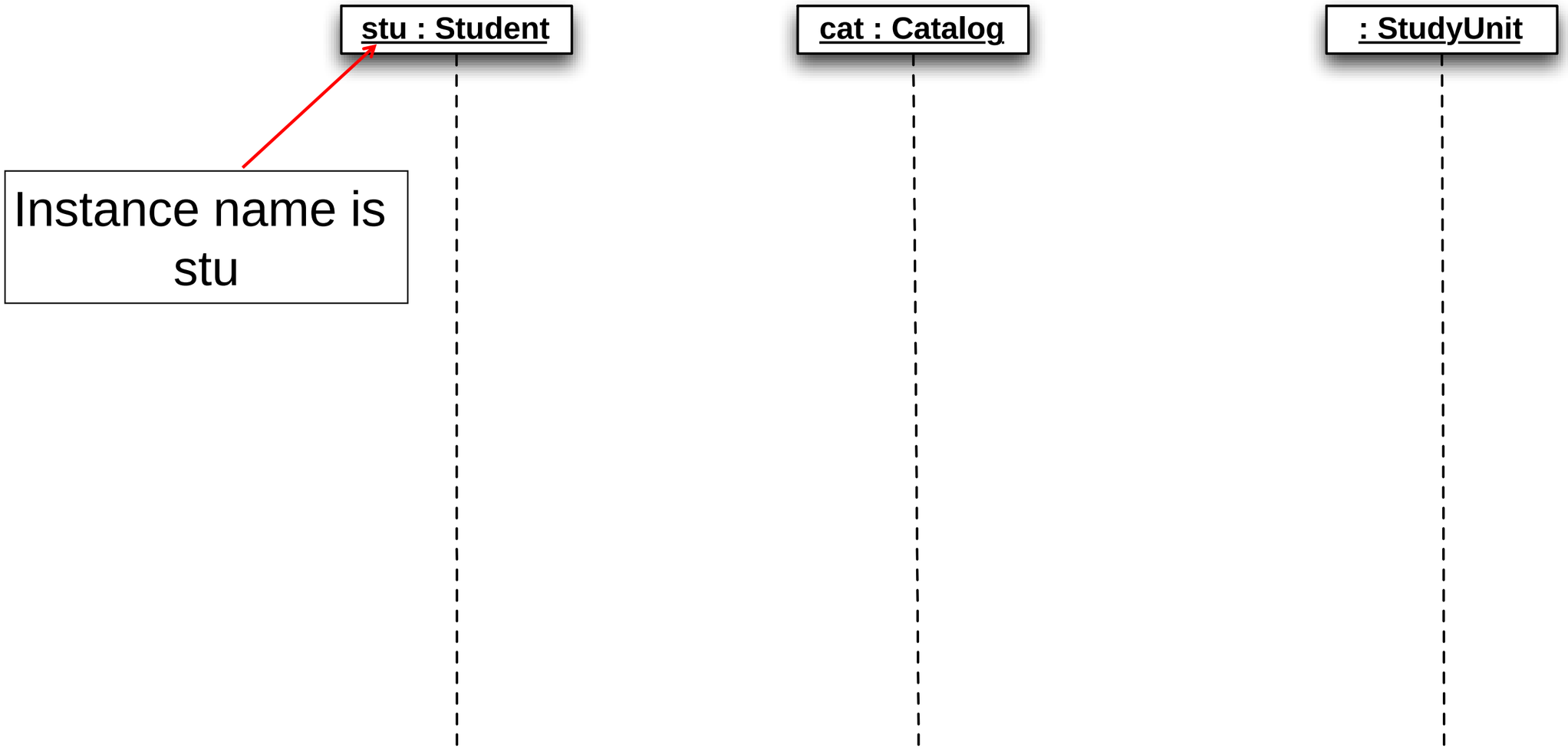
- write object's name if it clarifies the diagram



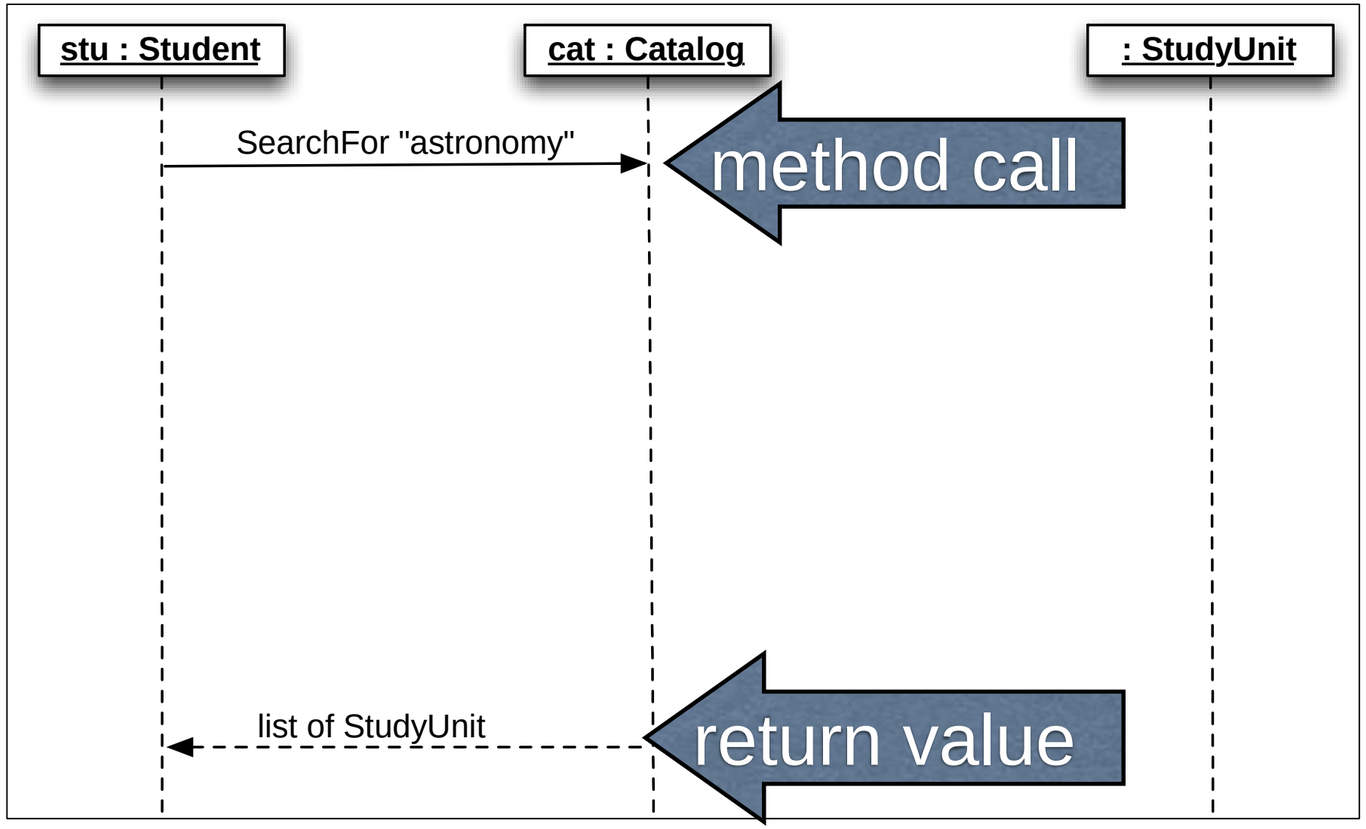
Name syntax: <objectname>:<classname>

Resource: http://csis.pace.edu/~marchese/CS389/L9/Sequence%20Diagram%20Tutorial.pdf

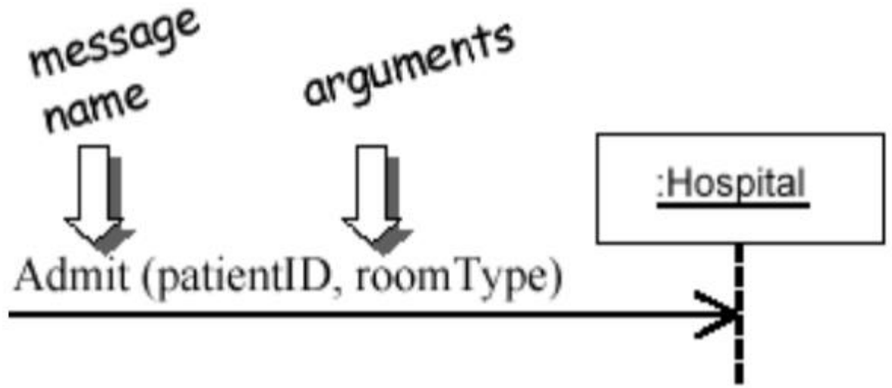
Life lines define the existence of objects



Messages are passed between objects

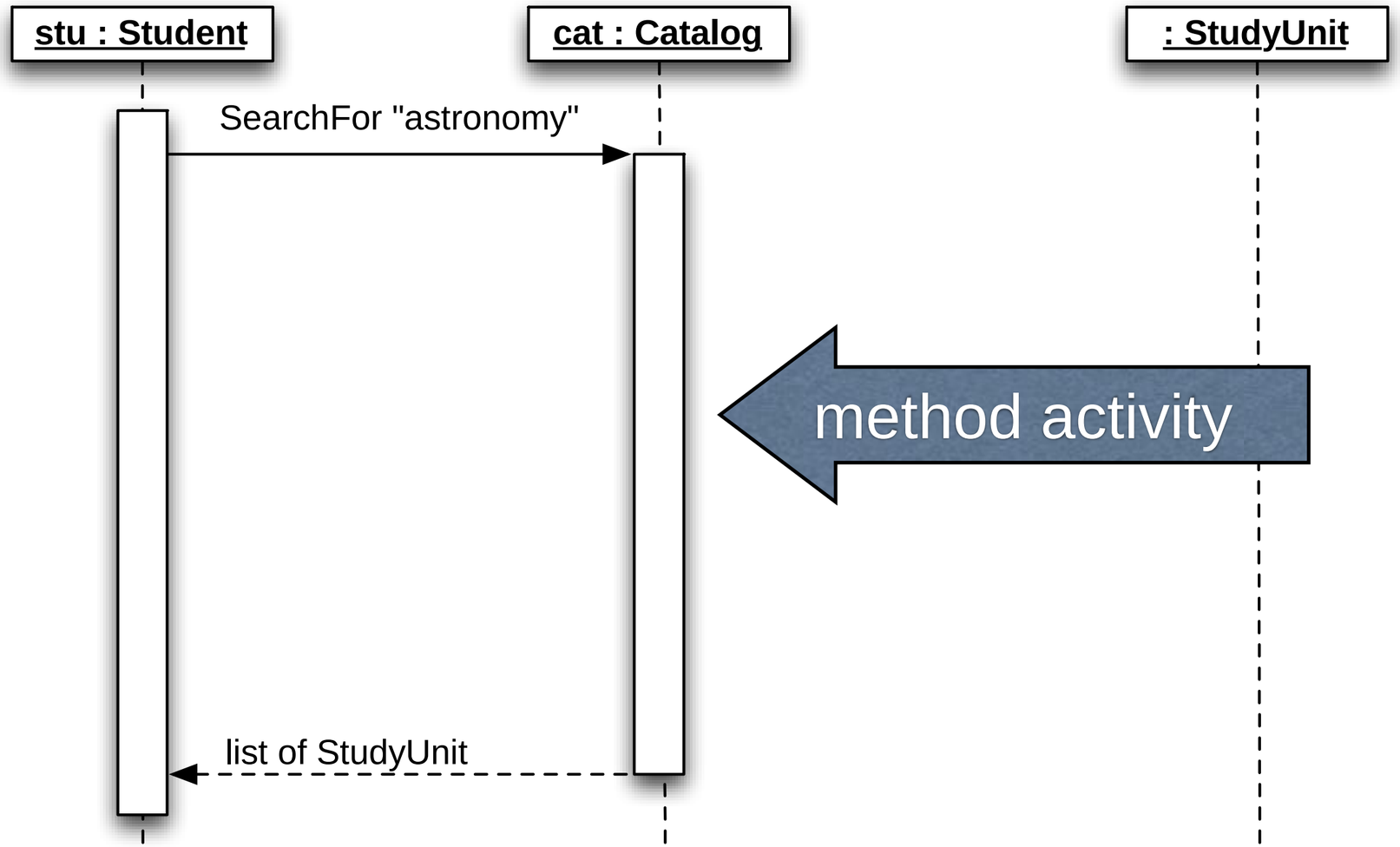


- **messages (method calls) indicated by arrow to other object**
- **write message name and arguments above arrow**



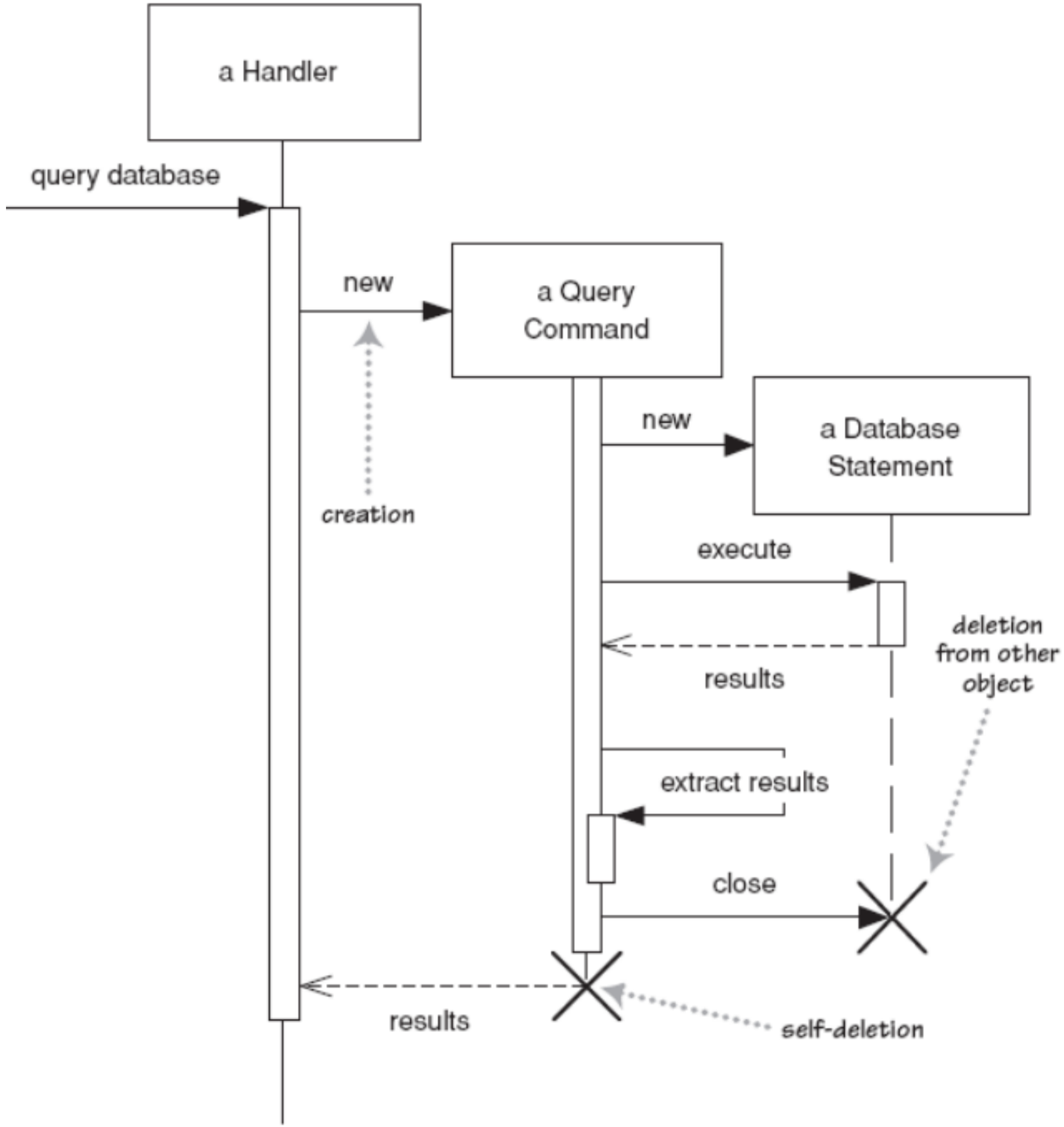
Resource: http://csis.pace.edu/~marchese/CS389/L9/Sequence%20Diagram%20Tutorial.pdf

Activity is represented by open boxes



Lifetime of objects

- **creation**: arrow with 'new' written above it
  - notice that an object created after the start of the scenario appears lower than the others
- **deletion**: an X at bottom of object's lifeline

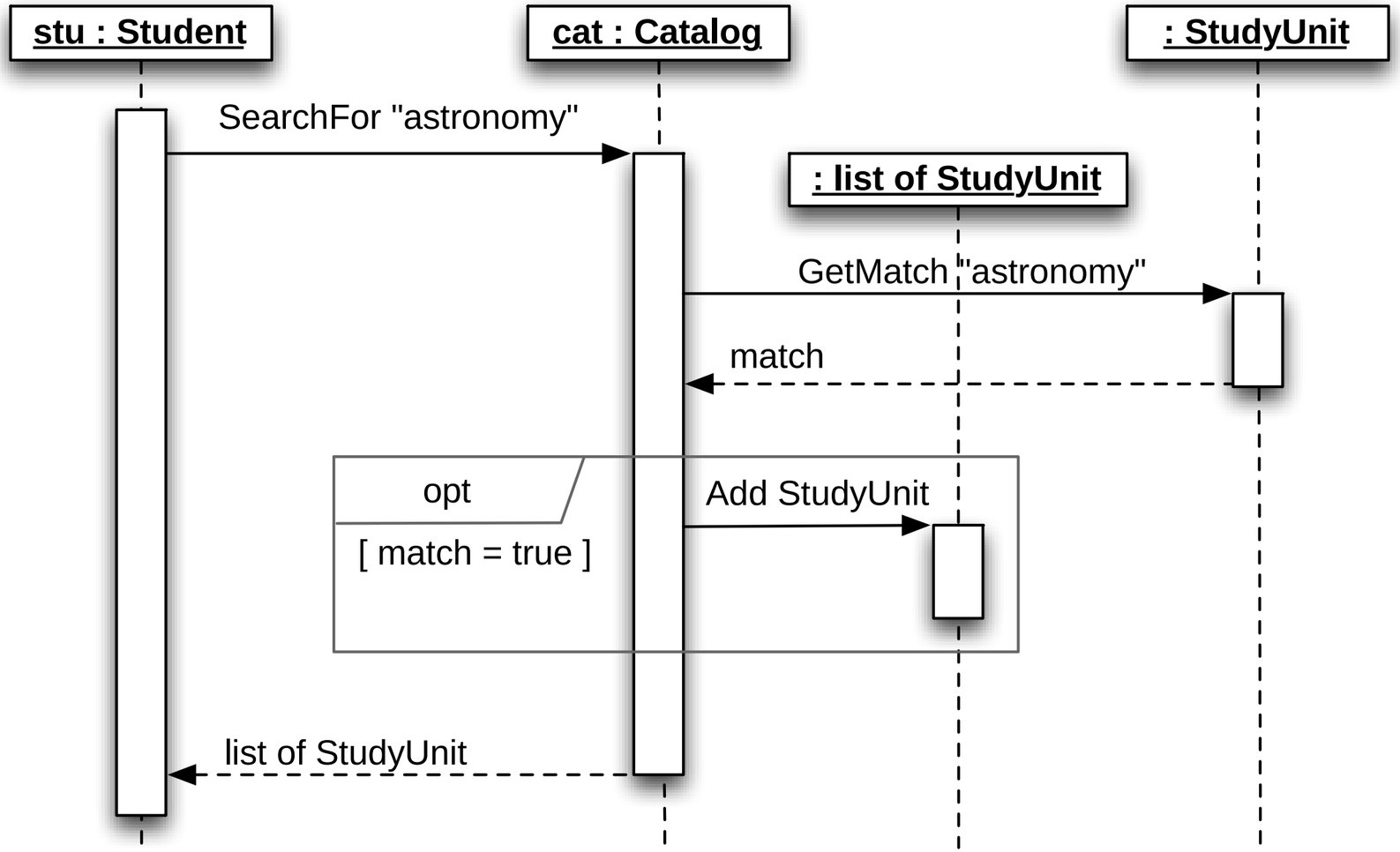


Resource: http://csis.pace.edu/~marchese/CS389/L9/Sequence%20Diagram%20Tutorial.pdf

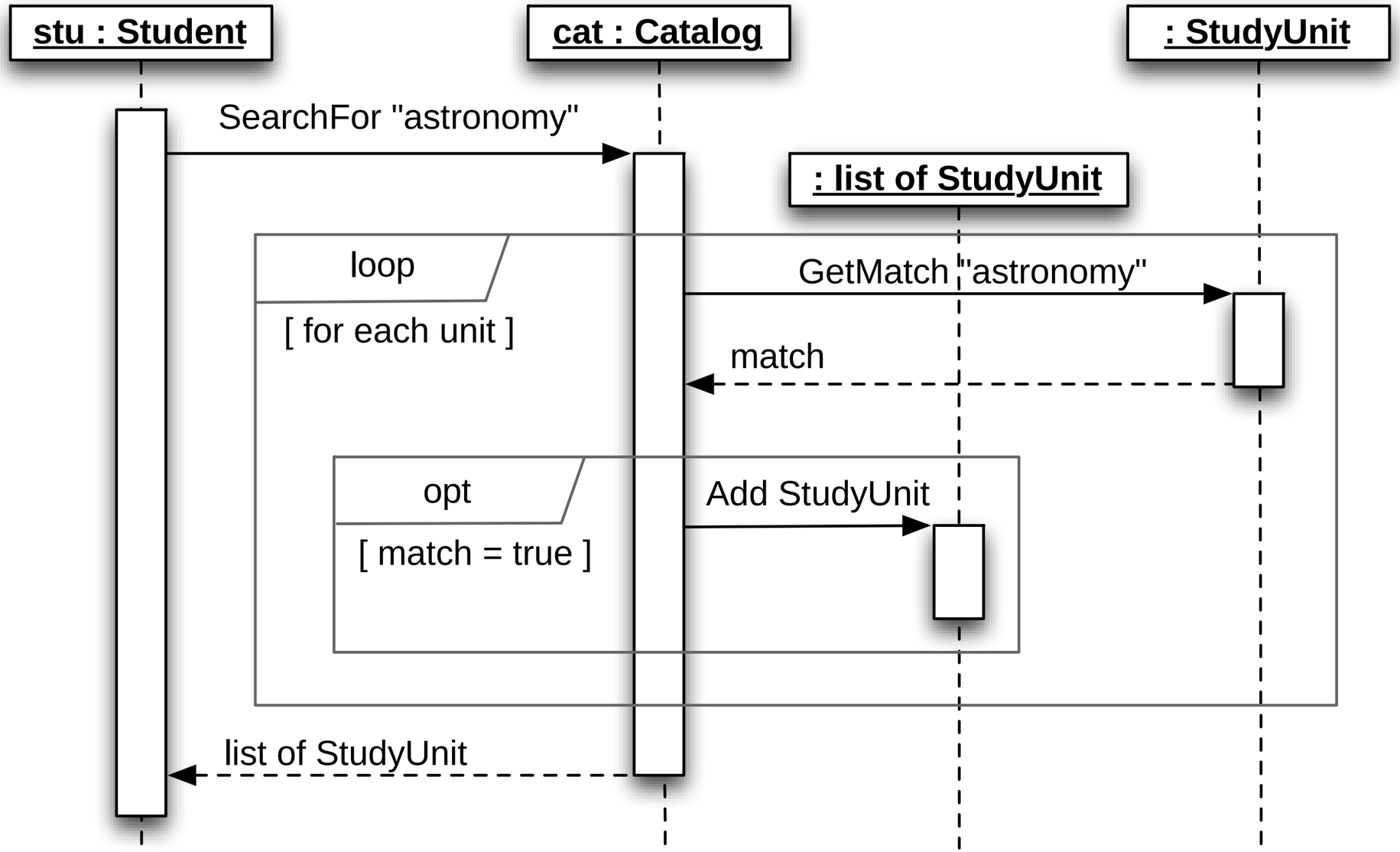
Control logic is described using ***combination fragments***

Options

- represents a **choice** of behavior
- [if condition]

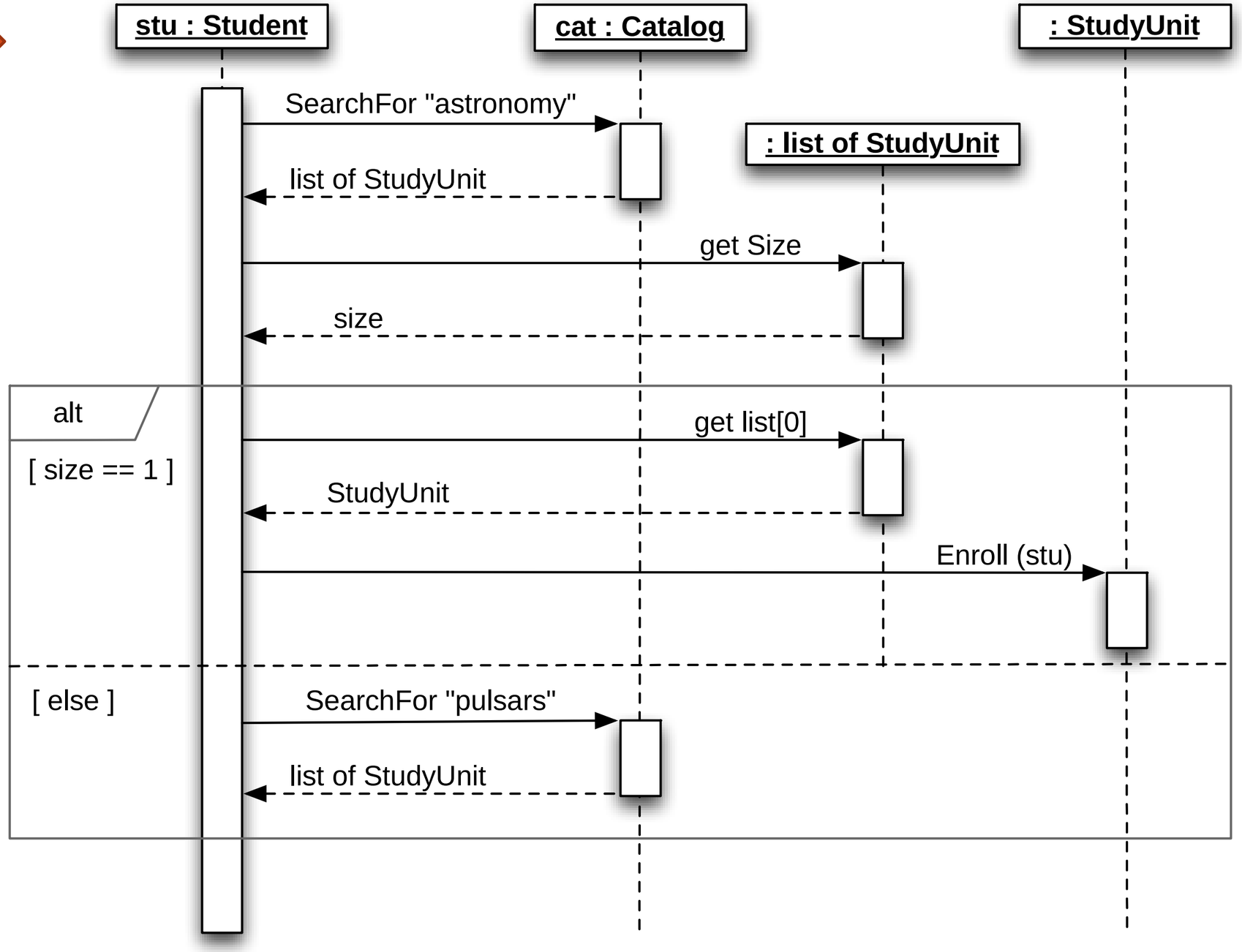


Loops


stu : Student
cat : Catalog
: StudyUnit
SearchFor "astronomy"
: list of StudyUnit
loop
[for each unit]
GetMatch "astronomy"
match
opt
[match = true]
Add StudyUnit
list of StudyUnit

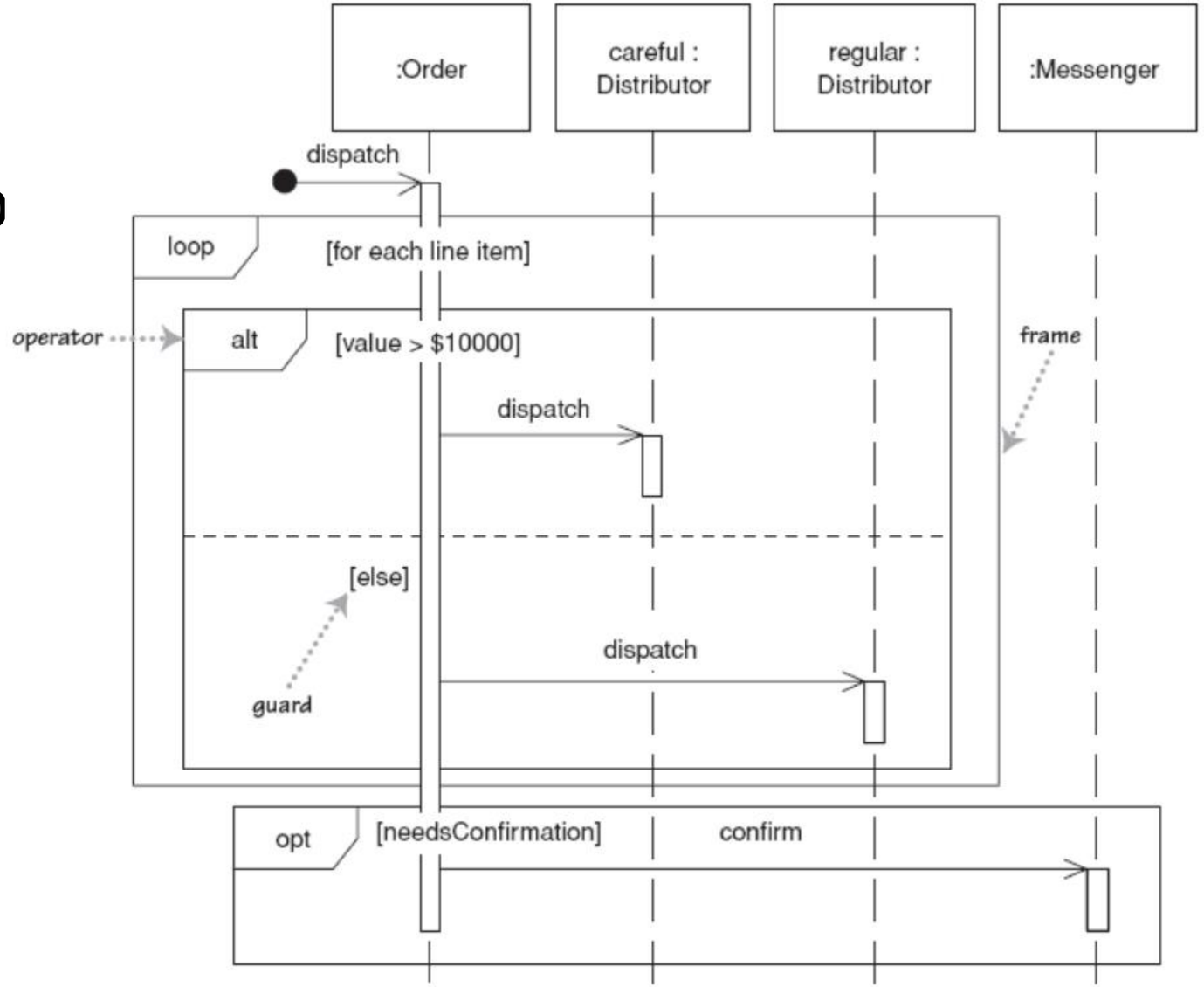
Alternatives

- represents a **choice** or alternatives of behavior
- one of the choices will be chosen
- [if-else condition]



Summary on Selection and Loops

- **frame:** box around part of diagram to indicate `if` or loop
  - `If` -> (opt) [condition]
  - `if/else` -> (alt) [condition], separated by horizontal dashed line
  - loop -> (loop) [condition or items to loop over]



Resource: http://csis.pace.edu/~marchese/CS389/L9/Sequence%20Diagram%20Tutorial.pdf

Solutions to complex problems require multiple iterations of design and discussion

The Unified Modeling Language provides a visual language for communicating design

The Unified Modeling Language can be understood by stakeholders

Communication
through UML Diagrams
can save time and effort

This Week's Tasks

Credit Task 1: System Modelling

Distinction Task 1: Custom Program UML Class Diagram